НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

«До захисту допущено»

В..Завідувача кафедри

_____ О.Л. Тимощук

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки 6.040303 Системний аналіз

на тему: «Система прийняття рішень для навігації в середовищах з неповною інформацією**»**

Виконав:
студент IV курсу, групи КА-53
Титаренко Андрій Миколайович _____

Керівник:
професор, д.ф.-м.н.,
Касьянов П. О. _____

Консультант з економічного розділу:
доцент, к.е.н.,
Шевчук О. А. _____

Консультант з нормоконтролю:
доцент, к.т.н.,
Коваленко А. Є. _____

Рецензент:

_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки (програма професійного спрямування) – 6.040303

Системний аналіз (Системи і методи прийняття рішень)

ЗАТВЕРДЖУЮ
В.О.Завідувача кафедри
_____ О.Л. Тимощук
«___»_____20__ р.

ЗАВДАННЯ

на дипломну роботу студенту

Титаренку Андрію Миколайовичу

1. Тема роботи «Система прийняття рішень для навігації в середовищах з неповною інформацією», керівник роботи Касьянов Павло Олегович, д.ф.-м.н., професор, затверджені наказом по університету від «___»_____ 2019 р. №_____

2. Термін подання студентом роботи

3. Вихідні дані до роботи _____

_____

4. Зміст роботи _____

_____

_____

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) _____

_____

_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Шевчук О.А. доцент		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка

Студент А. М. Титаренко

Керівник роботи П. О. Касьянов

РЕФЕРАТ

Дипломна робота: 123 с., 41 рис., 6 табл., 2 додатки, 49 джерел.

ВІЗУАЛЬНА НАВІГАЦІЯ, ГЛИБОКЕ НАВЧАННЯ, МАРКОВСЬКИЙ ПРОЦЕС ПРИЙНЯТТЯ РІШЕНЬ, НАБЛИЖЕНА ОПТИМІЗАЦІЯ СТРАТЕГІЇ, НАВЧАННЯ З ПІДКРІПЛЕННЯМ

Об’єктом дослідження є задача візуальної навігації. Предметом дослідження є застосування методів глибокого навчання з підкріпленням для вирішення задачі візуальної навігації.

Метою дослідження є розробка та навчання системи прийняття рішень для візуальної навігації на основі глибокого навчання з підкріпленням.

Дослідження ґрунтується на наукових статтях та інших матеріалах закордонних конференцій та архівів в галузі комп’ютерного зору, машинного навчання, глибокого навчання та глибокого навчання з підкріпленням.

В ході дипломної роботи розроблено та натреновано систему прийняття рішень, що з високими показниками успішності знаходить коротші шляхи до заданих цільових точок в сценах як із тренувальної, так і з валідаційної вибірок, демонструючи здатність до узагальнення.

Програмна реалізація моделі та інфраструктури навчання написана мовою Python, з використанням фреймворку PyTorch та симулятора HabitatSim.

ABSTRACT

Thesis work: 123 pp., 41 fig., 6 tabl., 2 app., 49 sources.

DEEP LEARNING, MARKOV DECISION PROCESS, PROXIMAL POLICY OPTIMIZATION, REINFORCEMENT LEARNING, VISUAL NAVIGATION

The object of the research is visual navigation task. The subject of study is application of deep reinforcement learning methods to visual navigation task.

The aim of study is the development and training of decision-making system for visual navigation based on deep reinforcement learning.

This research uses scientific papers and other materials of foreign conferences and achieves in field of computer vision, machine learning, deep learning and deep reinforcement learning.

During the work on thesis a decision-making system was implemented and trained. This system seeks for the shortest path to goal location with quantified success on scenes from both training and validation dataset splits, demonstrating its generalization ability to previously unseen data.

The implementation of model and training infrastructure was written using Python programing language, PyTorch – framework for dealing differentiable computation graphs, and HabitatSim simulator.

ЗМІСТ

ВСТУП 8
ПОСТАНОВКА ЗАДАЧІ 11
РОЗДІЛ 1 ЗАДАЧА ВІЗУАЛЬНОЇ НАВІГАЦІЇ ТА ПІДХОДИ ДО ЇЇ ВИРІШЕННЯ 12
1.1 Опис задач візуальної навігації 12
1.2 Класифікація задач навігації. 12
1.3 Актуальність задач 14
1.4 Метрики якості алгоритмів 17
1.5 Класичні методи 18
1.6 Задача навігації як Марковський процес прийняття рішень 21
1.6.1 Відомості про Марковські процеси прийняття рішень 21
1.6.2 Поняття стратегії 23
1.6.3 Задача навігації, як МППР з неповною інформацією 25
1.6.4 Обґрунтування складності задачі навігації 26
1.7 Висновки 28
РОЗДІЛ 2 МЕТОДИ ГЛИБОКОГО НАВЧАННЯ З ПІДКРІПЛЕННЯМ 29
2.1 Основні поняття навчання з підкріпленням 29
2.1.1 Задача навчання з підкріпленням 29
2.1.2 Стратегії та їх апроксимація 31
2.1.3 Функції вартості та кумулятивна винагорода 33
2.2 Таксономія алгоритмів глибокого навчання з підкріпленням 37
2.2.1 Алгоритми з моделями середовища 38
2.2.2 Алгоритми без моделей середовища 39
2.3 Навчання імітацією 41
2.4 Алгоритми градієнту стратегії 44
2.4.1 Виведення градієнту стратегії 44
2.4.2 Зниження дисперсії градієнту 47
2.4.2.1 Причинність 47
2.4.2.2 Базова вартість 48

2.5 Алгоритми типу Актор-Критик 49
2.5.1 Оцінювання переваги 49
2.5.2 Алгоритм Актор-Критик 51
2.6 Алгоритм Наближеної оптимізації стратегії 52
2.6.1 Проблема повторного використання прикладів 52
2.6.2 Наближена оптимізація стратегії 54
2.7 Методи вирішення проблеми дослідження 56
2.7.1 Дилема дослідження та експлуатування 56
2.7.2 Метод підрахунків 58
2.7.3 Навчання засноване на допитливості 59
2.8 Використання представлень середнього рівня 60
2.9 Висновки 62
РОЗДІЛ 3 ВИРІШЕННЯ ЗАДАЧІ ВІЗУАЛЬНОЇ НАВІГАЦІЇ МЕТОДАМИ ГЛИБОКОГО НАВЧАННЯ З ПІДКРІПЛЕННЯМ 64
3.1 Симулятор та середовище 64
3.1.1 Постановка задачі 64
3.1.2 Симулятор та дані 64
3.2 Запропонована модель для навчання 65
3.3 Використані методи вирішення проблеми дослідження 66
3.4 Структура реалізації програми для моделювання 68
3.5 Висновки 70
РОЗДІЛ 4 РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ ТА ЇХ АНАЛІЗ 71
4.1 Криві навчання 71
4.2 Основні джерела помилок 73
4.3 Висновки 75
РОЗДІЛ 5 ФУНКЦІОНАЛЬНО – ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ 76
5.1 Постановка задачі техніко-економічного аналізу 77
5.2 Обґрунтування функцій програмного продукту 77
5.3 Варіанти реалізації основних функцій 78

5.4 Обґрунтування системи параметрів ПП 80
5.4.1 Опис параметрів 80
5.4.2 Кількісна оцінка параметрів 81
5.4.3 Аналіз експертного оцінювання параметрів 82
5.5 Аналіз рівня якості варіантів реалізації функцій 86
5.6 Економічний аналіз варіантів розробки ПП 88
5.7 Вибір кращого варіанта ПП техніко-економічного рівня 94
5.8 Висновки 95
ВИСНОВКИ 97
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 98
ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО ПРОДУКТУ 103
ДОДАТОК Б ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДЛЯ ДОПОВІДІ 115

ВСТУП

Вже майже століття перед людством стоять проблеми робототехніки та штучного інтелекту, в вирішенні яких наука і техніка поступово покращують свої результати. До недавніх пір задачі комп'ютерного зору, прийняття рішень та управління вирішувалися з використовуванням апріорних знань в моделях та алгоритмах. Так, для задач класифікації [1 - 3] використовувалися рукотворні ознаки зображення, а для задач навігації - методи відтворення тривимірних поверхонь заснованих на проективній геометрії [4]. Протягом останнього десятиріччя відбувся інтенсивний розвиток глибокого навчання, причиною якого став розвиток обчислювальної техніки. Незалежні від доменної області, методи глибокого навчання перевершили найкращі алгоритми та методи з багатьох предметних областей таких, як комп'ютерний зір (зокрема в задачах класифікації зображень [5 - 7], детекції об'єктів [8, 9], семантичної сегментації [10], бінарної сегментації [11], генерування зображень [12] та інших), розпізнавання та генерування мови [13, 14], задач обробки природної мови. Крім того, як універсальні нелінійні апроксиматори функцій [15] нейронні мережі застосовуються і в задачах навчання з підкріпленням (керування, оптимізації, оптимізації недиференційованих функцій та інших). Такі моделі наразі успішно використовуються для багатьох ігор (нарди [16], шахи, шахи Го, Шогі [17, 18], Dota, StarCraft2), управління ресурсами та робототехніки. Останній напрямок зокрема виділяє багато підзадач керування роботами як на мікро- так і на макрорівні.

Об'єктом досліджень в даній роботі є задача візуальної навігації та планування в умовах, коли агент отримує зображення з камери низької якості та/або дані з дальномірів. В її рамках проводиться аналіз та порівняння існуючих методів навігації та планування, на основі яких розробляється власний метод з використанням навчання з підкріпленням. Таким чином, предметом дослідження є застосування методів глибокого навчання з підкріпленням для вирішення задачі візуальної навігації. Основною метою цієї роботи є розробка та навчання системи

прийняття рішень для візуальної навігації. Результати цієї роботи можуть бути застосовані для програмування навігаційної компоненти фізичного робота при використанні методів доменної адаптації.

В першому розділі розглядається задача візуальної навігації, її різновиди, класичні методи їх вирішення, а також визначені та описані Марковські процеси прийняття рішень, на термінологію яких досить природньо лягає задача навігації.

В другому розділі надається огляд і детальний опис методів глибокого навчання з підкріпленням, їх класифікація, основні сильні сторони і недоліки, а також методи, що дозволяють дещо позбавитись останніх.

В третьому розділі описується рішення, запропоноване для вирішення задачі візуальної навігації з використанням глибокого навчання з підкріпленням, архітектура та деталі програмної реалізації моделі та інфраструктури для її навчання.

В четвертому розділі розглядаються результати експериментів, наводиться їх аналіз, кількісне та якісне порівняння, аналіз процесу навчання та розбір основних помилок, що допускає найкраща модель.

ПОСТАНОВКА ЗАДАЧІ

1. Розробити систему для навчання та підтримки вибраних алгоритмів глибокого навчання з підкріпленням
2. Навчити агент, який на кожному кроці за візуальним сигналом (Зображення з камери та карта глибин) та інформацією про положення цілі (кут та відстань) приймав би рішення щодо руху, які б приводили до знаходження цілі за коротший час.
3. Упевнитися в здатності агенту до пошуку в середовищах, відсутніх в наборі для тренування.

РОЗДІЛ 1 ЗАДАЧА ВІЗУАЛЬНОЇ НАВІГАЦІЇ ТА ПІДХОДИ ДО ЇЇ ВИРІШЕННЯ

1.1 Опис задач візуальної навігації

В цій роботі під задачею навігації будемо розуміти задачу пошуку оптимальної послідовності дій, що призводить до попадання агента із початкового положення в кінцеве, що задається будь-яким зі способів. При цьому, на кожному кроці агент отримує інформацію про навколишнє середовище. Ця інформація може буди повною або неповною.

Повна інформація – це та, що повністю описує стан агента разом з навколишнім середовищем. В задачі навігації це може бути карта із відміченим положенням агента на ній. Для вирішення проблеми навігації із повною інформацією майже завжди можна застосувати один з класичних алгоритмів пошуку, таких як A*, алгоритм Дейкстри, та їх модифікації тощо. Проте в реальних умовах таку мапу дуже важко або неможливо побудувати, а тому найчастіше ми маємо лише неповну інформацію.

Неповна інформація в задачах пошуку зазвичай являє собою дані з сенсорів на роботі або деяких статичних конструкціях, що входять до навколишньої середи. Такі сенсори умовно можна поділити на дві категорії: ті, що дають інформацію про відстані або положення об’єктів у просторі і ті, що дають візуальну інформацію. До першої категорії відносять лідари, лазерні дальноміри, ехолокатори, датчики руху та інші. До другої належать здебільшого прилади фото та відео зйомки такі як камери та фотоапарати. Для задач навігації, що мають неповну інформацію використовують адаптивні алгоритми, які досліджують середовище одночасно з пошуком шляху.

1.2 Класифікація задач навігації.

Задачі навігації розрізнюють за декількома ознаками.

- за способом задання кінцевого стану:
    a) Точка в просторі

    Задачею агенту є пошук шляху від початкової точки до кінцевої точки, що задається вектором в кожен момент часу.

    b) Об’єкт

    В цьому випадку задачею є пошук об’єкта, що задається словом або фразою на природній мові. Наприклад, «холодильник», або «червоний стілець». Задача вважається виконаною, коли агент підходить до об’єкта на невелику відстань, або просто фіксує цей об’єкт в полі свого зору.

    c) Місцевість

    Спосіб схожий на «Об’єкт», але в цьому випадку задається назва приміщення або місцевості, в яку агенту треба прийти для завершення задачі. Наприклад, «кухня» або «їдальня».

    У другому та третьому випадку, агенту можуть знадобитися апріорні знання щодо того, що таке «кухня» або «дерев’яний стіл». Зазвичай необхідні представлення забезпечуються за допомогою попередньо натренованого класифікатора.

- за режимом дослідження місцевості.
    a) Без попереднього дослідження

    В цьому випадку агенту не дають часу на дослідження місцевості. Він повинен будувати шлях базуючись на знаннях про середовище, здобутих саме під час пошуку.

    b) За попередньо записаним дослідженням середовища

    Агенту надаються попередньо записані дані про середовище. Наприклад, наявні відео обльоту місцевості, або деякі фотознімки кімнат, тощо. Детально підходи до того, як витягти корисну інформацію та використати її в таких випадках розглядаються в [19]

    c) З виділенням часу на дослідження середовища

Агенту дається час на дослідження місцевості перед початком виконання задач.

1.3 Актуальність задач

Тепер, маючи класифікацію задач, наведемо приклади важливих застосувань алгоритмів, що вирішують їх та класифікуємо кожен з них.

Ефективні мобільні операції в 3-вимірному просторі є важливою темою дослідження в Штучному Інтелекті. В сучасному світі поширюється використання роботів на автономних систем, що призначені замінити або полегшити людську працю, зробити її більш безпечною та прискорити її. Адаптивні автономні алгоритми пошуку шляху дуже важливі в багатьох застосуваннях робототехніки. Наведемо декілька прикладів таких застосувань в різних сферах людської діяльності:

- сфера безпеки

  Для працівників сфер безпеки пожежної, рятівної та поліцейської служб дуже важливо забезпечити власну безпеку при виконанні завдань. Вони самостійно проникають в небезпечні середовища: квартири, будинки, приміщення різного типу. В розвинених країнах в їх розпорядженні вже великий час знаходяться спеціальні роботи різного ступеня автономності. Проте більшість таких роботів керується оператором, адже до недавніх пір використовування адаптивних алгоритмів потребувало дуже дорогих обчислювальних ресурсів. Робота оператора не завжди є ефективною, адже він не завжди може адекватно інтерпретувати всі можливі сенсори та оптимально приймати рішення. Особливо це прослідкується в умовах поганої видимості. Крім того, для такого робота потрібен хороший зв’язок для забезпечення можливості віддаленого контролю та спеціальний оператор, що унеможливлює повсюдне їх використання. Тепер, коли недороге апаратне забезпечення

дозволяє використовувати адаптивні технології, їх застосування може допомогти серйозно розвинути можливості автономних помічників.

– домашні помічники

Сьогодні багато хто використовує домашніх помічників. Роботи пилососи, голосові асистенти, системи розумних домів стають більш розумними кожен день, демонструючи успіхи в розумінні людської мови, навігації в домі, контролі показників приміщення тощо. Зі сторони навігації більшість з них влаштовані досить просто і опираються на класичні алгоритми побудування мап та планування із їх використанням, мова про які піде у другому розділі. Проте більш складні задачі такі роботі виконують не дуже успішно, наприклад коли потребується більш серйозне знання навколишнього світу, властивостей речей, контролю та розпізнавання образів за мультимодальних даними. Таким чином, попри необхідність та велику кількість зусиль, роботи, що доглядають за людьми з обмеженими фізичними можливостями поки що не є дуже поширеними через свою велику вартість, а також недовіру людства до роботів. Проте, наприклад, в Японії станом на 2015 рік на [20] (рисунок 1.1).

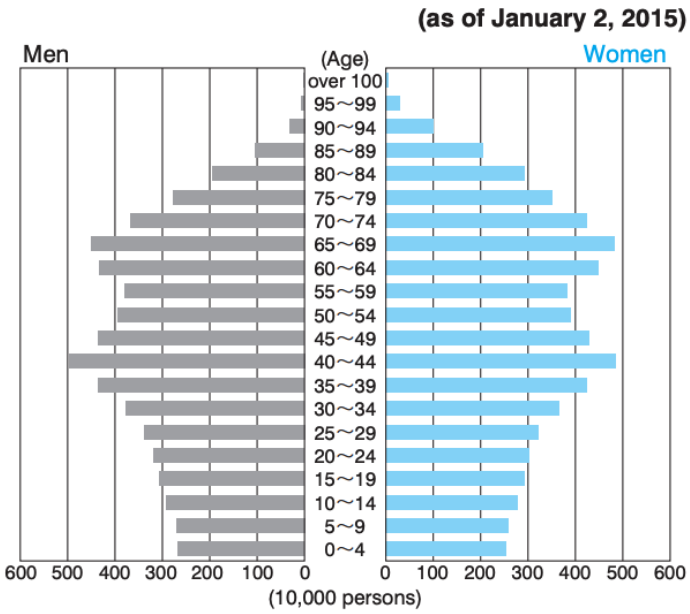


Рисунок 1.1 - Діаграма зі звіту Японської Асоціації Медсестер (Japanese Nursing Association) щодо складу населення в Японії на момент січня 2015 року

відсоток населення, що дожило до 65 років та більше становить 26.5% і за прогнозами підніметься до 39.9%. Тобто на дві людини похилого віку прийдеться три молоді або середнього віку. Таким чином неймовірно зросте потреба в догляді за ними, яку буде неможливо задовільнити при використанні тільки людських ресурсів. На даний момент, навіть просте прохання принести склянку води для штучного інтелекту є складною задачею, тому необхідність розвивати відповідні технології є очевидною.

- як важлива підсистема автономних роботів

  Насправді дуже велика кількість застосувань роботів включає в себе задачу навігації, як одну з підзадач. Сюди можна включити будь-яких роботів, які переміщуються по замкненим приміщенням. Тому вирішення задачі навігації може зробити можливим побудову більш складних індустріальних, ремонтних роботів та автоматизованих систем, що можуть замістити велику кількість професій. Також сюди можна віднести мікробіомедичних роботів, що призначені для переміщення по внутрішнім органам пацієнтів в пошуках заданих аномалій. Попри те, що такі розміри унеможливлюють розміщення обчислювального інтелекту на борту, наявна можливість проведення обчислювально складних операцій, використовуючи зовнішні ресурси.

- багатовимірні простори для пошуку

  Багато задач можна переформулювати в термінах пошуку, тому вирішення задачі навігації також розвиває інструментарій для вирішення більш абстрактних задач пошуку, таких як пошук в просторі зображень, графів та навіть функцій із параметричного сімейства. Більшість цих задач мають застосування в машинному навчанні та управлінні.

Таким чином, задачі навігації з обмеженою інформацією є актуальними на сьогодні, як головна задача, яку агент розв’язує, та як одна з задач, що входять до складу виконуваних роботом при роботі.

1.4 Метрики якості алгоритмів

Для адаптивних алгоритмів пошуку дуже важливою є оцінка якості їх роботи. Для цього використовують різні метрики оптимальності. Всі вони оцінюють алгоритм базуючись на шляхах, що він зазвичай знаходить. Для їх обчислення вибирають велику кількість епізодів використовування алгоритму. Епізод являє собою набір, що складається з середовища, заданої початкової точки та кінцевої множини точок.

Найпоширенішою з метрик вважають так звану SPL-метрику (Success Path Length) [21] що виглядає наступним чином:

$$SPL = \frac{1}{N}\sum_{i=1}^{N} S_i \frac{l_i}{\max(p_i, l_i)},$$

де $N -$ кількість епізодів;

l_i – довжина шляху, що був знайдений алгоритмом;

p_i – геодезична відстань між початковою та кінцевою точками (довжина найкоротшого шляху);

S_i – індикатор успішного знаходження шляху.

Наприклад, якщо агент в середньому знаходить шлях в 50% випадків та цей шлях є вдвічі довшим за геодезичну відстань, тобто $l_i = 2p_i$, то $SPL = 0.25$, а якщо алгоритм завжди знаходить найкоротший шлях, то $SPL = 1.0$. Таким чином значення цієї метрики лежать від 0 до 1, а більші значення відповідають кращим алгоритмам.

1.5 Класичні методи

В цьому підрозділі будемо вважати, що на вхід алгоритму подається карта глибин, що являє собою відображення точок, спроектованих на площину камери, на відстань від площини камери до них (рисунок 1.2).

Рисунок 1.2 - RGB зображення з камери та карта глибин, що йому відповідає. Зображення взяті з набору даних Gibson

Класичні методи, що наразі є найпоширенішими, являють собою ітеративне планування дій на декілька кроків вперед. Кожна ітерація складається з трьох етапів: оцінювання власного положення в просторі, побудова карти та етапу планування.

На першому етапі алгоритм повинен оцінити своє місцезнаходження та орієнтацію в просторі. Зазвичай на початку роботи місцезнаходження робота встановлюється в центр мапи, яку він будує протягом своєї роботи. Після кожного виконання запланованих дій, положення в просторі перераховується. Більшість моделей дискретизує дії, які може виконувати робот. Наприклад, рух вперед на 0.25 метра, повороти вліво та вправо на 10 градусів та деякі їх комбінації. Таким чином, кожного разу нове положення перераховується із врахуванням виконаних дій. Такий підхід пропонується, наприклад, в [44]. Також слід урахувати зіткнення із перешкодами, яке здатне дещо змінити траекторію руху.

На другому етапі доповнюється загальна тривимірна мапа. Її побудова складається з двох кроків: зворотнього проектування точок з карти глибин на тривимірний простір та саме доповнення мапи.

1. Внутрішні (intrinsics) параметри камери подаються у вигляді матриці (camera matrix)

Для тривимірного простору матриця параметрів записується як:

$$C = \begin{matrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{matrix},$$

де $f_x,\ f_y$ – власні параметри камери;

c_x, c_y – координати центра площини камери відносно кута.

Рівняння проектування із точки з тривимірного простору на може бути записано як:

$$\begin{matrix} x \\ y \\ w \end{matrix} = \begin{matrix} f_x & 0 & c_x & X \\ 0 & f_y & c_y & Y \\ 0 & 0 & 1 & Z \end{matrix},$$

де $\begin{matrix} x \\ y \\ w \end{matrix}$ – 2-вимірний вектор в однорідних координатах;

w – відповідає за масштаб.

В нашому випадку w – відоме, адже

$$w = Z$$

Тепер для знаходження $\begin{matrix} X \\ Y \\ Z \end{matrix}$ можемо скористатися формулою:

$$\begin{matrix} X \\ Y \\ Z \end{matrix} = \begin{matrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{matrix}^{-1} \begin{matrix} x \\ y \\ Z \end{matrix}$$

Для спрощення обчислень, отримаємо вирази для X, Y:

$$X = \frac{x - c_x Z}{f_x},$$
$$Y = \frac{y - c_y Z}{f_y}$$

Таким чином, за картою глибин ми знайшли множину точок в 3-вимірному просторі, що їй відповідають. Зазначимо також, що координати цих точок відкладаються від центра зображення, тобто після зворотного проектування необхідно перевести отримані вектори в загальну систему координат з центром в початковій точці.

2. Після чергового зворотного проектування, отримані точки дискретизуються та додаються до існуючої множини, що складає мапу.

В реальних додатках дані з сенсорів мають шумову складову і задача побудови мапи включає ймовірнісну складову, тобто треба визначати розподіл:

$$P\ m_t, x_t\ o_{1:t}),$$

де o_t – спостереження агенту;

x_t – розташування агенту;

m_t – мапа середовища.

На кожному кроці правило Баєса дає можливість оновлювати мапу-розподіл та розташування агенту при отриманні нових даних:

$$P\ x_t\ o_{1:t}, m_t) = \sum_{m_{t-1}} P\ o_t\ x_t, m_t) \sum_{x_{t-1}} P(x_t|x_{t-1})\, P(x_{t-1}|m_t, o_{1:t-1})/Z,$$

де Z – нормалізуючий коефіцієнт.

$$P\ m_t\ o_{1:t}, x_t) = \sum_{x_t} \sum_{m_t} P\ m_t\ x_t, m_{t-1}, o_t\ \ P(m_{t-1}, x_t | m_{t-1}, o_{1:t-1})$$

Ця задача вирішується за допомогою ЕМ-алгоритму, розширеного фільтру Калмана, фільтром часток. Найпоширенішим методом є Пакетне Корегування (Bundle Adjustment), який також дуже популярний в задачі калібрування параметрів камер.

На третьому етапі відбувається планування наступної серії рухів. Для цього можуть використовуватися різні алгоритми пошуку шляхів, наприклад A* (з деякими модифікаціями), алгоритм Дейкстри, а також Fast Marching Method, що наразі є найпоширенішим в даній задачі.

1.6 Задача навігації як Марковський процес прийняття рішень

1.6.1 Відомості про Марковські процеси прийняття рішень

Марковським процесом прийняття рішень (МППР) називають набір з 4 множин $(S, A, T, R\ \ )$, де:

1. $S -$ множина станів.
2. $A -$ множина дій.
3. $P_a\ s, s'\ = \Pr\ s_{t+1} = s'\ s_t = s, a_t = a)$ – ймовірності переходів, в залежності від дії. Такі $P_a\ s, s'$ формують оператор переходів T.

Позначимо:

$$\mu_{t,j} = p\ s_t = j$$

$$\xi_{t,k} = p\ a_t = k$$

$$T_{i,j,k} = p\ s_{t+1} = i | s_t = j, a_t = k$$

Тоді:

$$\mu_{t,i} = \sum_{j,k} T_{i,j,k}\mu_{t,j}\xi_{t,k},$$

4. $R(s,a)$ функція підкріплення. Визначає кількість нагороди, що отримує агент, здійснюючи дію a, попри те, що знаходиться в стані s.

Марковським цей процес називається через те, що для них характерна марковська властивість, яку Девід Сільвер зазвичай формулює, як те що майбутнє залежить від сьогодення і не залежить від минулого, тобто стан в поточний момент часу є достатньою статистикою (рисунок 1.3).

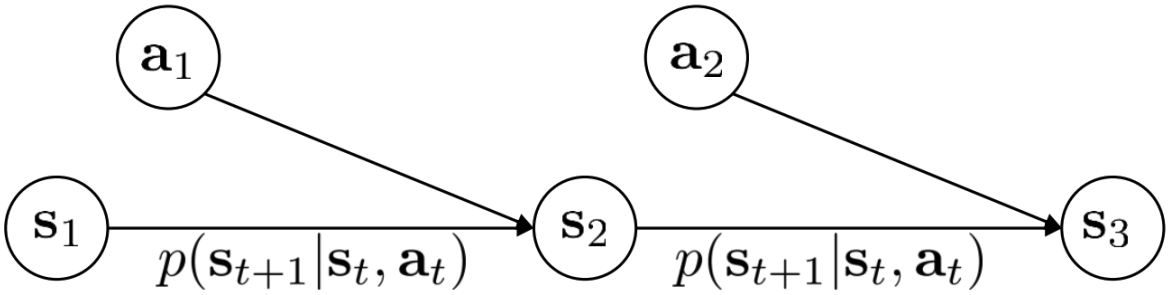


Рисунок 1.3 - Графічна модель, що описує МППР

Крім того, МППР може бути з повною і неповною інформацією (fully, partially observed). Наявність неповної інформації означає, що агент не знає всієї інформації про стан s_t. Його вхідні спостереження позначаються як o_t. Прикладом такого процесу можна навести 2-вимірний лабіринт, в якому агентові видно тільки навколишні клітинки (рисунок 1.4).

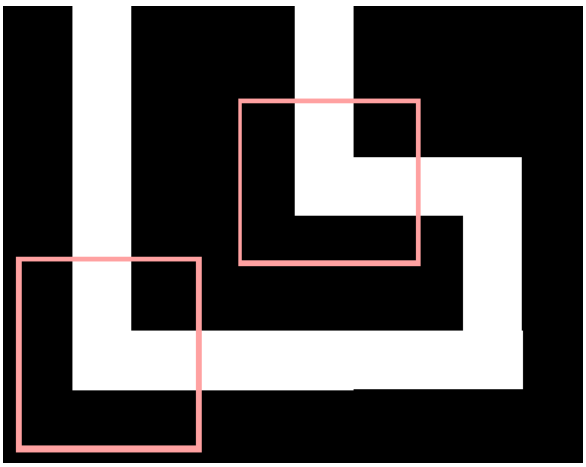


Рисунок 1.4 - МППР з неповною інформацією. Тут відмічені регіони співпадають як спостереження агенту, якщо той знаходиться в середині

В таких МППР можлива ситуація, коли спостереження агенту може бути однаковим попри те, що стани різні. Ця та інші проблеми унеможливлюють використання багатьох алгоритмів.

Більш формально (рисунок 1.5), МППР з неповною інформацією є набором $(S, A, O, T, \mathrm{E}, R)$:

1. S – множина станів.
2. A – множина дій.
3. O – множина спостережень, що надається агенту навколишнім середовищем.
4. T – оператор переходів, що визначається тим же чином, що і у випадку МППР.
5. E – ймовірність емісії спостережень $p(o_t|s_t)$, де $o_t \in O$.
6. R – функція підкріплення, $R: S \times A \to \mathbb{R}$.

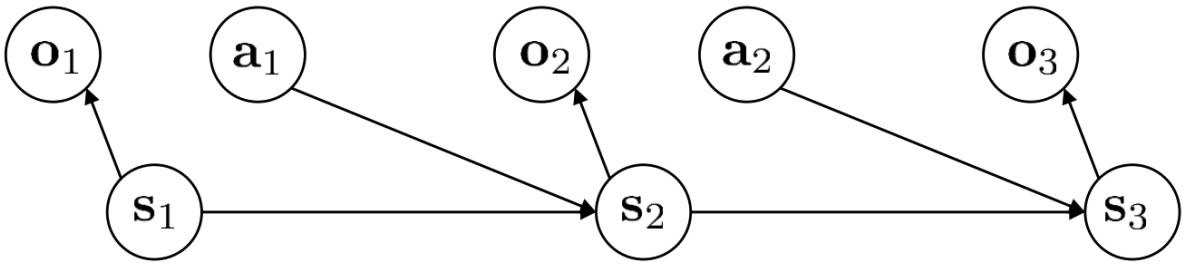


Рисунок 1.5 - МППР із неповною інформацією у вигляді графічної моделі

1.6.2 Поняття стратегії

Стратегією (policy) називають функцію π, що за станом s визначає розподіл $p\ a\ \ s$) дій, які приймає агент. Стратегія зветься детермінованою, якщо розподіл включає ймовірності 1 та 0. Стратегія, разом з оператором переходів визначають розподіл траєкторій (траєкторія – послідовність станів та дій $\tau = (s_1, a_1, \dots, s_T, a_T)$).

$$p_\pi\ \tau\ = p\ s_1 \prod_{t=1}^{T} \pi\ \ a_t\ s_t\ p\ s_{t+1}\ s_t, a_t)$$

В рамках МППР агенту потрібно знайти стратегію $\pi^*\ s$, (для МППР з неповною інформацією $\pi^*\ o$), який максимізує очікувану винагороду:

$$\pi^* = argmax_\pi \mathbb{E}_{\tau \sim p_\theta(\tau)} \sum_t R(s_t, a_t)$$

Таким чином, процес можна зобразити у вигляді наступної графічної моделі (рисунок 1.6).

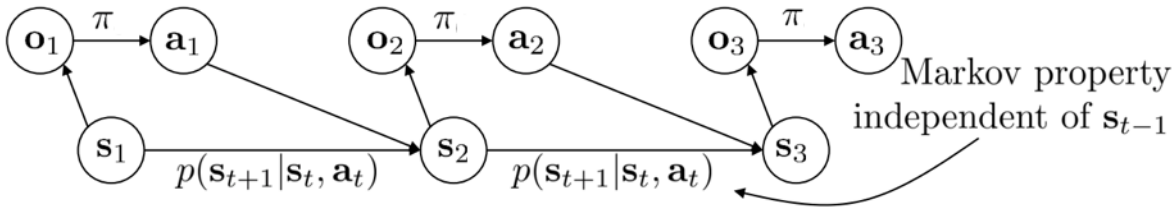


Рисунок 1.6 - МППР в загальному вигляді. На рисунку зображена графічна модель, що демонструє структуру МППР

Для Марковських процесів прийняття рішень із скінченними просторами дій та станів доведена теорема про існування оптимальної марковської інформаційної стратегії, що максимізує цільовий функціонал (очікувану кумулятивну винагороду) [49].

1.6.3 Задача навігації, як МППР з неповною інформацією

Можна показати, що задача навігації підпадає під опис МППР з неповною інформацією:

1. $O-$ показники з сенсорів та інформація про шукану точку.
2. S – місцезнаходження та положення агенту.
3. $A-$ множина можливих. Найчастіше до неї належать: рух прямо на деяку відстань, повороти направо або наліво навколо власної осі, а також дія, якою агент сигналізує про завершення завдання.
4. В якості функції підкріплення можна вибрати будь-яку, проте найчастіше вона має наступну форму:

$$R\ s_t, a_t\ = R_g + R_c + R_p + R_l,$$

де $R_g = dist_{geo}(s_t, s_{t-1})$;

$dist_{geo} -$ геодезична відстань;

$R_c -$ від'ємна винагорода, що надається агенту при зіткненнях з перешкодами;

$R_p -$ від'ємна винагорода, що надається на кожному кроці для заохочення агенту вибирати більш короткі шляхи;

$R_l -$ нагорода, яку агент отримує, коли успішно знаходить шлях.

В цій роботі ми будемо розглядати спрощені абстрактні сенсори, які надають RGBD зображення, тобто зображення із 3 кольоровими каналами і 1 каналом глибини, що являє собою карту глибини.

Задачі МППР вирішують в тому числі за допомогою навчання з підкріпленням. Для даної задачі, в якості параметричного сімейства стратегій, доречним вибором являються нейронні мережі (рисунок 1.7)

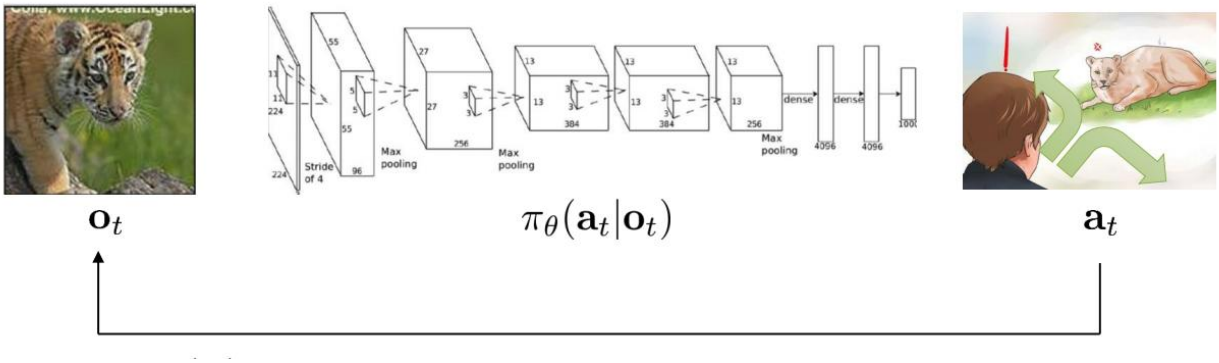


Рисунок 1.7 - Нейронна мережа в якості апроксиматора стратегії

Більш детально алгоритми навчання з підкріпленням, що використовують глибоке навчання описуються в Розділі 2.

1.6.4 Обґрунтування складності задачі навігації

Основними джерелами складності в даній задачі є складність простору спостережень, складність дослідження середовища та наявність неповної інформації про поточний стан агенту в середовищі.

Простір спостережень в цій роботі являє собою простір дуже великої розмірності, оскільки агент спостерігає зображення та мапу глибин. Для роботи з такими складними даними потрібно використовувати методи зниження розмірності або екстрактори ознак (feature extractors) засновані на згорткових нейронних мережах.

Крім того, до візуальних спостережень додається інформація щодо цілі пошуку, що може також являти собою текст на природній мові, що потребує обробки для подальшого використання моделлю прийняття рішень.

Іншою проблемою вирішення задачі навігації є складність дослідження середовища. Наприклад, якщо агент шукає шлях до близької за евклідовою відстанню точки, що лежить за стіною та, проте, є дуже віддаленою за

геодезичною відстанню, адже між початковим та кінцевим станом лежить множина складно з’єднаних приміщень (рисунок 1.8)

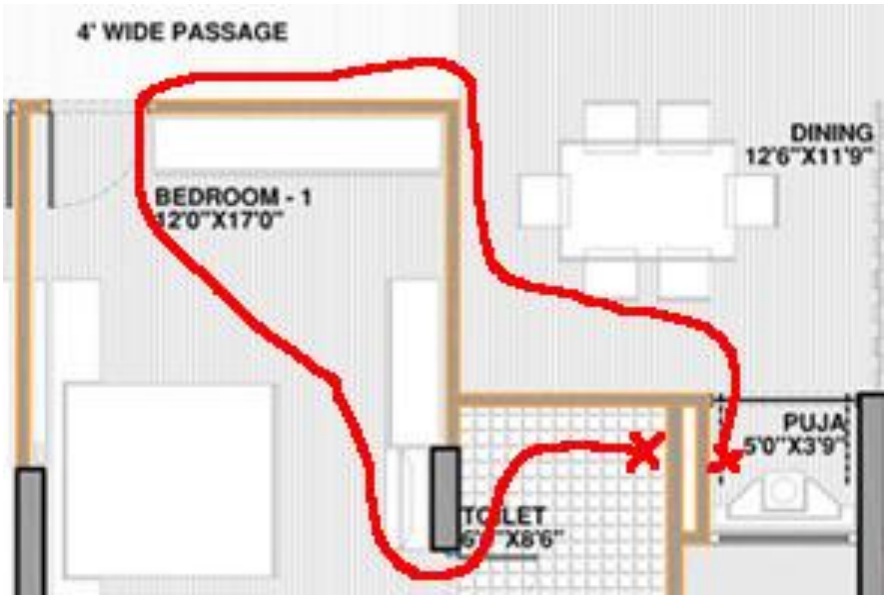


Рисунок 1.8 - Епізод, в якому початкова і кінцева точки лежать близько з точки зору евклідової відстані та далеко за геодезичною відстанню

Для адаптивних алгоритмів ефективний процес дослідження є дуже важливим, адже для оптимізації шляху агент повинен досягти цілі в процесі дослідження, а на ранніх епохах навчання таких агентів, їх стратегії схожі на гаусівські процеси.

Неповна інформація також ускладнює життя агенту, адже йому потрібно враховувати свої спостереження включаючи ті, що агент отримав у минулому. Без цього агент може збоїти коли бачить дуже схожі зображення та об’єкти на них.

1.7 Висновки

В цьому розділі була розглянута задача навігації та класичні методи її вирішення, а також оглянуті Марковські процеси прийняття рішень. Було дане визначення і класифікація задач навігації, показана їх актуальність та важливість їх дослідження для робототехніки та її застосувань в різних сферах людської діяльності таких, як сфера безпеки, медицини, підвищення рівня життя на інших.

В рамках огляду класичних методів була розглянута типова схема класичних алгоритмів для побудови мапи і планування дій агенту, таких як SLAM. Були описані основні етапи роботи таких алгоритмів та методи, що виконують певні підзадачі в таких системах, наприклад, Fast Marching Method (FMM) для планування дій та метод зворотного проектування мапи глибин.

Під час огляду МППР було дано їх визначення, надані деякі їх властивості. Було дане визначення МППР з неповною інформацією, описані деякі рівняння, що характеризують динаміку цих процесів, дане визначення стратегії та сформульована основна задача в рамках МППР. Була переформульована задача навігації в термінах МППР з неповною інформацією, що дає можливість застосувати методи їх вирішення для задач навігації. Такі методи розглянуті в розділі 2. В кінці розділу була дана характеристика складності задачі навігації, основні її джерела та проблеми, ними породжені.

Основна увага була приділена протиставленню класичних підходів до задач навігації підходам, в рамках яких задача розглядається як МППР з неповною інформацією.

РОЗДІЛ 2 МЕТОДИ ГЛИБОКОГО НАВЧАННЯ З ПІДКРІПЛЕННЯМ

2.1 Основні поняття навчання з підкріпленням

2.1.1 Задача навчання з підкріпленням

Навчання з підкріпленням, на ряду із навчанням з та без учителя, є окремою парадигмою машинного навчання. Маючи багато спільного із останніми, навчання з підкріпленням формалізує ідею того, що якщо винагороджувати або наказувати агента за певний тип поведінки, то він більш імовірно почне повторювати або уникати цю поведінку в майбутньому. Проте, його не можна віднести до жодного з них, адже існує низка відмінностей.

1. На відміну від алгоритмів з учителем, в навчанні з підкріпленням модель не має чітких «правильних» відповідей. На винагороду можуть впливати минулі рішення агенту, динаміка середовища та інше. Винагорода може буте дуже розрідженою, коли як, наприклад, в шахах агент може отримувати одиницю підкріплення лише наприкінці партії, не даючи підказки про правильність окремих дій. В навчанні з підкріпленням алгоритму надається лише інформація про правильність траєкторії агенту, а інформація про те, які дії слід вибирати залишається невідомою
2. В навчанні без учителя відповідей не надається або зовсім, або майже зовсім (так зване напівавтоматичне навчання). Алгоритми, що мають справу з подібними випадками зазвичай є генеративними або алгоритмами кластеризації, які загалом можна охарактеризувати як ті, що шукають структуру або розподіл даних та використовують її.

В схемі навчання з підкріпленням приймають участь агент і середовище. На кожній ітерації середовище генерую стан та надає агенту спостереження. Агент, в свою чергу, вибирає одну з можливих дій, на що отримує винагороду та наступне спостереження. Рисунок 2.1 ілюструє дану схему взаємодії.

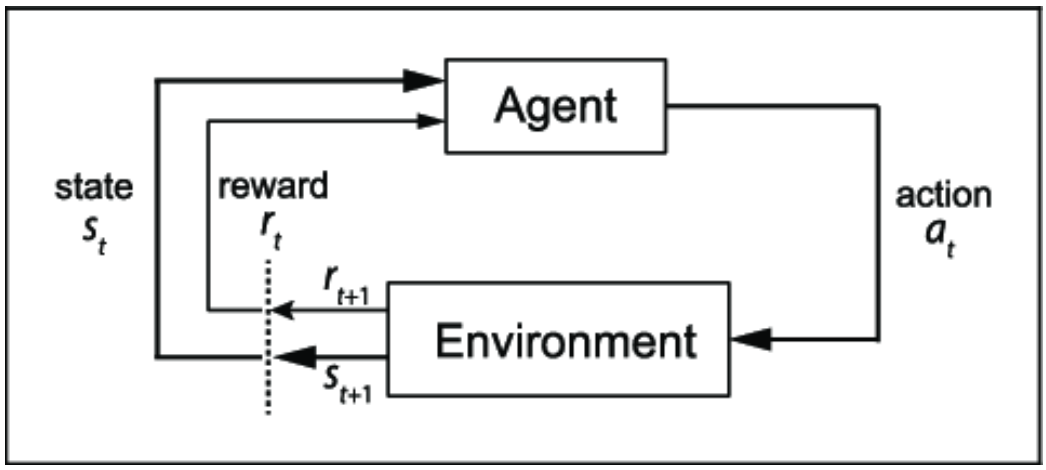


Рисунок 2.1 - Схема взаємодії середовища з агентом

Стан, що генерується середовищем містить повну інформацію про внутрішній стан світу. Спостереження, які отримує агент, в свою чергу можуть містити лише частину цієї інформації. Наприклад, при грі в «Покер» агент отримує інформацію про карти, що він має на руках, а також про ті, що лежать на столі. Однак повний стан системи включає в себе також карти всіх інших гравців, які агент не бачить. Таким чином спостереження не дають повної інформації про стан середовища, що їх генерує.

Множини всіх спостережень і дій можуть бути як дискретними (розміщення фігур на шаховій дошці, ігрові карти, що бачить агент), так і неперервними (координати рухомих частин скелета, сила, яку агент застосовує до певної частини механізму), мати різну розмірність та структуру.

Крім того, в даній схемі повинна виконуватися марківська властивість. Зауважимо, що під таку модель підпадає безліч реальних випадків, і, хоча спочатку може здаватися, що марківська властивість не виконується, тобто:

$$p\ s_{t+1}\ s_0, s_1, \dots, s_t\ \neq p(s_{t+1}|s_t)$$

В багатьох ситуаціях можливе переозначення станів s_t, як, наприклад,

$$s_t = (s_t, s_{t-1}, \dots, s_{t-k})$$

Насправді, таке переозначення можливе завжди – достатньо лише покласти $k = t$, тобто станом нового процесу буде вся передісторія станів. На практиці, такий випадок не є доцільним, адже таке накопичення станів є неефективним с точки зору обчислень та пам'яті. У випадках, коли $k \ll T$, такий прийом дозволяє ефективно використовувати весь арсенал алгоритмів, що вирішують МППР.

Неважко побачити, що більш формально, проілюстрований процес являє собою Марковський процес прийняття рішень. Тобто задача, яку вирішує агент:

$$\tau = (s_1, a_1, \dots, s_T, a_T)),$$

$$p_\pi\ \tau\ = p\ s_1 \prod_{t=1}^{T} \pi\ a_t\ s_t\ p\ s_{t+1}\ s_t, a_t),$$

$$\pi^* = argmax_\pi \mathbb{E}_{\tau \sim p_\theta(\tau)} \sum_t R(s_t, a_t)$$

2.1.2 Стратегії та їх апроксимація

Загалом, стратегії можна розділити на детерміновані та стохастичні. Детермінованою стратегією називають невипадкову функцію:

$$\mu: o_t \longmapsto a_t,$$

що співставляє спостереженню $o_t \in O$ дію $a_t \in A$.

Слід зазначити, що у випадку, коли не виконується марковська властивість, ця функція може також інкапсулювати інформацію про більшу кількість станів в одному, для перетворення процесу прийняття рішень на Марковський, наприклад, за допомогою введення внутрішнього стану h_t:

$$\mu: (o_t, h_{t-1}) \longmapsto\ \ a_t, h_t$$

Стохастичною стратегією називають випадкову функцію:

$$a_t \sim \pi(\cdot\,|s_t)$$

На практиці вона реалізовується, як:

$$\pi\ a_t\ s_t\ = p(a_t|s_t)$$

Тобто приймаючи стан, або спостереження МППР, вона повертає умовний розподіл дій із простора дій, з якого потім випадковим чином генерується дія. У випадках, коли простір дій неперервний, розподіл дій часто вважають діагональним нормальним, з центром в $\pi\ a_t\ s_t$, та діагональною коваріаційною матрицею, що зазвичай апроксимується незалежно.

Стратегія може мати різні представлення. В багатьох класичних алгоритмах навчання з підкріпленням, де має місце її наближення, ця функція часто представлялася в табличному вигляді. Однак, коли розмірність простору дій, або простору станів є великою, такий підхід перестає бути ефективним через те, що така таблиця займає багато місця в пам'яті.

В глибокому навчанні з підкріпленням, стратегію наближують за допомогою параметричних апроксиматорів, таких як нейронні мережі. В такому випадку, функцію позначають π_θ, або μ_θ у випадку, коли вона є детермінованою, де θ – набір параметрів моделі. Задача, що стоїть перед алгоритмом формулюється наступним чином:

$$p_\pi\ \tau\ = p\ s_1 \prod_{t=1}^{T} \pi_\theta\ a_t\ s_t\ p\ s_{t+1}\ s_t, a_t),$$

$$\theta^* = argmax_{\pi_\theta}\mathbb{E}_{\tau\sim p_\theta(\tau)} \sum_t R(s_t, a_t)$$

2.1.3 Функції вартості та кумулятивна винагорода

Позначимо $r_t = R\ s_t, a_t\ .$

Кумулятивною винагородою за скінченний час без дисконтування (finite-horizon undiscounted return) називають винагороду, що агент сумарно отримує протягом скінченної кількості кроків:

$$R\ \tau\ = \sum_{t=0}^{T} R(s_t, a_t)$$

Така винагорода має сенс, коли епізод є обмеженим за часом. Як приклад можна привести агент, що вчиться керувати робо-рукою для з'єднання двох деталей за обмежений час (рисунок 2.2).

Рисунок 2.2 - Робо-рука вчиться з'єднувати дві деталі за обмежений час

Дисконтованою кумулятивною винагородою за нескінченний час (infinite-horizon discounted return) називають винагороду що агент сумарно отримує протягом нескінченної кількості кроків:

$$R\ \tau\ = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t),$$

де γ – коефіцієнт дисконтування, $0 < \gamma < 1$.

Окрім гарантування збіжності ряду (за деяких цілком реалістичних допущень стосовно функції винагород), коефіцієнт дисконтування має ще декілька призначень. По-перше, він виконує функцію ваги винагороди. Для винагород, що були отримані раніше, ця вага буде більшою, таким чином їх вклад в кумулятивну винагороду буде більшим, ніж у більш віддалених. По-друге, його можна розглядати, як імовірність переходу в новий стан «смерті» МППР, діючи таким чином, для агенту більш привабливими є винагороди, що можливо отримати раніше. По-третє, зменшуючи значення доданків він зменшує загальну дисперсію кумулятивної винагороди, що є дуже важливим для алгоритмів, заснованих на градієнті стратегії (policy gradient), про які піде мова далі в цьому розділі.

Функцією вартості зветься функція $V^{\pi}\ s$, що являє собою очікувану кумулятивну винагороду за умови, що початковий (поточний) стан – s:

$$V^{\pi}\ s\ = \mathbb{E}_{\tau\sim\pi}[R\ \tau\ |s_0 = s]$$

Q-функцією зветься функція $Q^{\pi}\ s, a$, що являє собою очікувану кумулятивну винагороду за умови, що початковий (поточний) стан – s, а відповідна йому дія агенту – a.

$$Q^{\pi}\ s, a\ = \mathbb{E}_{\tau\sim\pi}[R\ \tau\ |s_0 = s, a_0 = a]$$

Оптимальною функцією вартості $V^{*}\ s$ називається функція вартості, що відповідає оптимальній стратегії π^{*}:

$$V^{*}\ s\ = \max_{\pi}(\mathbb{E}_{\tau\sim\pi}\ R\ \tau\ \ s_0 = s\ )$$

Оптимальною Q-функцією $Q^{*}\ s, a$ називається Q-функція, що відповідає оптимальній стратегії π^{*}:

$$Q^*\ s,a\ = \max_{\pi}(\mathbb{E}_{\tau\sim\pi}\ R\ \tau\ \ s_0 = s, a_0 = a\ )$$

Неважко показати, що ці функції зв'язані наступними рівняннями:

$$V^{\pi}\ s\ = \mathbb{E}_{a\sim\pi}\ Q^{\pi}\ s,a\ \ ,$$
$$V^*\ s\ = \max_{\text{a}}[Q^*\ s,a\ ]$$

Всі наведені вище функції є взаємоузгодженими. Цю залежність ілюструє наступне судження. Вартість поточного стану – це кумулятивна винагорода, яку очікують отримати за знаходження в цьому стані, плюс вартість стану, в який агент прямуватиме далі. Цей принцип виконується для функцій вартості та Q-функцій одночасно. Рівняння, що формалізують його звуться рівняннями Белмана.

Рівняння Белмана для функції вартості та Q-функції мають наступний вигляд:

$$V^{\pi}\ s\ = \mathbb{E}_{a\sim\pi\ \cdot\ s\ ,s'\sim P\ \cdot\ s,a}\ \ R\ s,a\ + \gamma V^{\pi}\ s'\ \ ,$$
$$Q^{\pi}\ s,a\ = \mathbb{E}_{s'\sim P\ \cdot\ s,a}\ \ R\ s,a\ + \gamma\mathbb{E}_{a'\sim\pi\ \cdot\ s}\ [Q^{\pi}\ s',a'\ ]\ ,$$

де $P\ \cdot\ s,a$ – імовірність розподіл наступних станів при умові знаходження в стані s та виборі дії a;

γ – коефіцієнт дисконтування, який можна вважати рівним 1 для випадку, коли кумулятивна нагорода не є дисконтованою.

Для оптимальної функції вартості та оптимальної Q-функції рівняння Белмана приймають вигляд:

$$V^*\ s\ = \max_{\text{a}}\mathbb{E}_{s'\sim P\ \cdot\ s,a}\ \ R\ s,a\ + \gamma V^*\ s'\ \ ,$$
$$Q^*\ s,a\ = \mathbb{E}_{s'\sim P\ \cdot\ s,a}\ \ R\ s,a\ + \gamma\max_{a'}[Q^*\ s',a'\ ]$$

Рівняння Белмана для оптимальних функцій можна використовувати для пошуку оптимальної стратегії. Для прикладу наведемо два відомих класичних алгоритми: Ітерація Вартостей (Value Iteration) та Ітерація Стратегій (Policy Iteration).

Ітерація Вартостей обчислює оптимальний стан функції вартостей ітеративно покращуючи оцінку $V(s)$. Спочатку алгоритм ініціалізує $V(s)$ випадковим чином. Далі він послідовно оновлює $Q(s,a)$ та $V(s)$ до моменту збіжності. Для МППР алгоритм Ітерації Вартостей гарантовано збігається до оптимальних значень. Псевдокод алгоритму зображений на рисунку 2.3.

Повторювати:

Для всіх $s \in S$

Для всіх $a \in A$

$$Q(s,a) \leftarrow \mathbb{E}[R|s,a] + \gamma \sum_{s' \in S} P(s'|s,a)V(s')$$

$$V(s) \leftarrow \max_a Q(s,a)$$

До збіжності $V(s)$

Рисунок 2.3 - Псевдокод алгоритму ітерації вартостей

На відміну від Ітерації Вартостей, який продовжує покращувати оцінку функції вартостей до її збіжності, Ітерація Стратегій працює до збіжності саме стратегії. Оскільки агенту треба оптимізувати саме стратегію, оптимальна стратегія може бути знайденим раніше, ніж функція вартостей. Алгоритм ІП у вигляді псевдокоду зображений на рисунку 2.4.

Ініціалізувати план π' довільним чином

Повторювати:

$\pi \leftarrow \pi'$

Обчислити вартості за допомогою π вирішуючи СЛАР:

$$V^{\pi}(s) = \mathbb{E}[R|s,\pi(s)] + \gamma \sum_{s' \in S} P\left(s'\middle|s,\pi(s)\right)V^{\pi}(s')$$

Покращуємо план для кожного стану:

$$\pi'(s) \leftarrow argmax_a(\mathbb{E}[R|s,a] + \gamma \sum_{s' \in S} P(s'|s,a)V^{\pi}(s'))$$

До збіжності π $(\pi = \pi')$

Рисунок 2.4 - Псевдокод алгоритму ітерації стратегій

Як вже зазначалося, обидва цих методи гарантовано збігаються в МППР та деяких його модифікаціях [22]. Проте, головною проблемою цих алгоритмів є те, що для представлення функцій вартості у табличному вигляді, що нечасто є можливим у реальних застосуваннях, де складність простору станів або дій може зробити такий підхід неефективним з обчислювальної точки зору. Крім того, ці алгоритми спираються на відомість динаміки переходів, тобто для їх роботи потрібно знати тензор переходів.

2.2 Таксономія алгоритмів глибокого навчання з підкріпленням

Загалом класифікацію алгоритмів глибокого навчання з підкріпленням можна зобразити у вигляді наступної схеми [23 - 32] (рисунок 2.5):

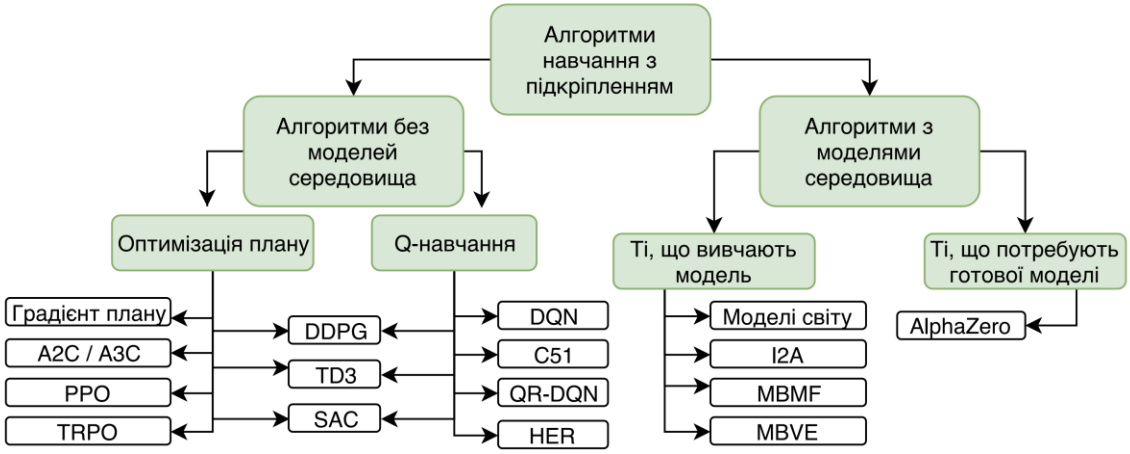


Рисунок 2.5 - Класифікація алгоритмів глибокого навчання з підкріпленням

Майже всі вони мають структуру, що показана на рисунку 2.6.

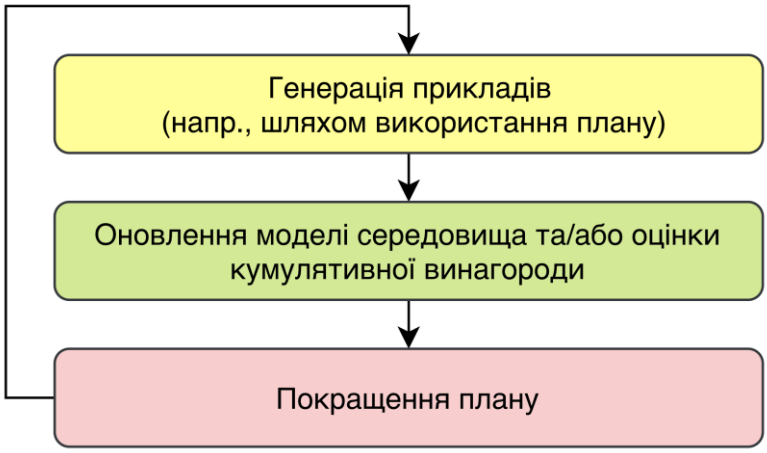


Рисунок 2.6 - Структура алгоритму глибокого навчання з підкріпленням

Алгоритм складається з трьох етапів, що послідовно виконуються до досягнення бажаного результату.

На першому етапі відбувається генерування нових прикладів. Під прикладом мається на увазі, наприклад, набір (s_t, R_t, a_t, s_{t+1}), що складається з поточного стану (або спостереження), наступного стану, винагороди, отриманої агентом та дії, що агент здійснив. Множину таких наборів можна отримати використовуючи стратегію в симуляції середовища (або в самому середовищі, якщо це можливо). Слід зазначити, що ця множина може складатися з різної кількості прикладів. Якщо в ній всього один кортеж, то алгоритм називають он-лайн алгоритмом. Крім того, деякі алгоритми для генерації завжди потрібують поточну версію стратегії (on-policy), в той час, як інші використовують поточну оптимальну версію стратегії (off-policy), що дозволяє повторно використовувати згенеровані приклади.

На другому етапі відбувається оновлення (fitting) додаткових моделей, які використовує алгоритм. Це може бути модель середовища, або модель, що оцінює вартості, тощо.

На третьому етапі стратегія покращується певним чином. Слід зазначити, що не всі алгоритми явно моделюють або функціонально наближують стратегію. Деякі приклади таких методів будуть надані в наступних підрозділах.

2.2.1 Алгоритми з моделями середовища

Одним з підходів у навчанні з підкріпленням передбачається сімейство алгоритмів, що використовує модель середовища (Model-based). Вони, в свою чергу, поділяються на ті, що вивчають модель під час загального навчання, та ті, що використовують вже існуючу модель, передбачену автором.

До другого класу належить відомий алгоритм AlphaZero [18], попередня версія якого, AlphaGo [17], відома перемогою чемпіону світу з Го Лі Седоля. AlphaGo використовувала пошук по Монте-Карло дереву, яке моделювало процес антагоністичної настільної гри Го. Простір спостережень цього алгоритму

включав надзвичайно велику кількість вручну підібраних показників (наприклад, кількість каменів, що знаходяться в певних ситуаціях, тощо). Сам же AlphaZero перевершив попередника, отримуючи лише стан дошки, в якості спостереження. Такий підхід дозволив використати цей алгоритм для багатьох інших, подібних Го, ігор, як шахи та шогі (японські шахи), суттєво перевершивши домінуючі в цих іграх підходи.

До першого класу належать алгоритми Моделі Світу (World Models), I2A (Imagination-Augmented Agents) [33], MBMF (Model-Based RL with Model-Free Fine-Tuning) [34], що належить і до безмодельних алгоритмів в деякій мірі також, та інші. Проте в цій роботі цей клас алгоритмів більш детально розглядатися не буде.

2.2.2 Алгоритми без моделей середовища

Альтернативним підходом в навчанні з підкріпленням є безмодельний (model-free). Його особливістю є те, що він не передбачає явну оцінку динаміки середовища, а концентрується лише на ітеративному покращенні стратегії. В цьому класі алгоритмів можна виділити два напрямки: оптимізація стратегії та Q-навчання.

В основі алгоритмів Q-навчання лежать дві основні ідеї:

1. При відомій Q-функції можливо вибирати наступну дію не маючи окремої моделі для стратегії:

$$\pi'(a_t|s_t) = \begin{matrix} 1, \text{якщо } a_t = argmax_{a_t} Q^{\pi}\ s_t, a_t \\ 0, \text{в інших випадках} \end{matrix}$$

Якщо Q^{π} відома (або оцінена с достатньою точністю), то a_t буде, як мінімум, не гіршою за будь яку $a_t \sim \pi(a_t|s_t)$. Крім того, якщо $Q^{\pi} = Q^{*}$, то отримана стратегія гарантовано буде оптимальною. Таким чином, випливає факт

того, що оптимізуючи Q^{π} можна знайти оптимальну стратегію, явно його не моделюючи.

2. Оптимальну Q-функцію можна отримати мінімізуючи, так звану, помилку Белмана:

$$\mathcal{E} = \frac{1}{2}\mathbb{E}_{s,a\sim\beta}\left[Q_\theta(s,a) - \left(R(s,a) + \gamma \max_{a'} Q_\theta(s',a')\right)\right],$$

де β - розподіл всіх можливих прикладів;

θ - набір параметрів моделі, яка наближує Q-функцію.

Якщо $\mathcal{E} = 0$, тоді $Q_\theta(s,a) = R(s,a) + \gamma \max_{a'} Q_\theta(s',a')$, тобто оптимальна Q-функція знайдена. Алгоритм представлений на рисунку 2.7.

Повторювати:
Генерація прикладів виду (s_i, a_i, s_{i+1}, r_i)
Повторити K разів:
$y_i \leftarrow R(s_i, a_i) + \gamma \max_{a_i'} Q_\theta(s_i', a_i')$
$\theta \leftarrow argmin_\theta \frac{1}{2}\sum_i \|Q_\theta(s_i, a_i) - y_i\|^2$
До збіжності

Рисунок 2.7 - Псевдокод алгоритму Q-навчання

Слід зазначити, що на відміну від аналогічного алгоритму для табличного представлення Q-функції, збіжність процедури мінімізації не є гарантованою.

Так відбувається через те, що оператор Белмана (дія, що виконується першою на кожній ітерації) є стиском за sup-нормою, в той час як оператор проектування на простір моделей (дія, що виконується другою на кожній ітерації) є стиском за L_2 нормою. Їх композиція стиском не є. В табличному випадку проектування не відбувається, а оскільки нерухомою точкою оператора Белмана (для Q-функцій) є оптимальна Q-функція, то алгоритм збігається. Проте, як зазначалося раніше, табличне представлення часто не є обчислювально досяжним,

а тому Q-навчання використовується із прийомами поліпшення збіжності, що є більш емпіричними.

Таким чином, алгоритми цього підходу не мають під собою твердого теоретичного обґрунтування. Їх популярність здебільшого зумовлена можливістю генерації прикладів без використання поточної стратегії, що суттєво прискорює процес навчання, покразуючи його ефективність. В даній роботі алгоритми даного типу використовуються лише для протиставлення їх ефективності запропонованому підходу.

До напрямку оптимізації стратегії відносять, наприклад, алгоритми [23 - 25], а [26, 28] включають в себе ідеї обох напрямків. Саме алгоритми цього сімейства здебільшого використовуються в цій роботі. Далі в розділі мова піде про них.

2.3 Навчання імітацією

Навчання імітацією (imitation learning) – підхід, в рамках якого стратегія вивчається в режимі навчання з учителем.

Найпростішим алгоритмом в цьому підході є клонування поведінки (behavior cloning). Основною ідеєю цього алгоритму є клонування стратегії умовного експерту. Експерт може бути представленим в будь-якому вигляді, наприклад, це може бути інша стратегія, класичний алгоритм без навчання або людина. Цей алгоритм потребує приклади виду (o_t, a_t), де o_t - спостереження, а a_t – дія. Ці приклади генеруються експертною стратегією. Далі, в рамках навчання з учителем, відбувається навчання стратегії на цих прикладах.

У випадку, коли стратегія представлена нейронною мережею, а простір дій є дискретним, навчання відбувається наступним чином:

$$\theta = argmin_\theta \; - \sum_{a_t, o_t \in D} \sum_{i}^{C} a_t^i \log(f\ h_\theta\ o_t\ _i) \ ,$$

де $h_\theta(o_t)$ – нейронна мережа з параметрами θ;

$f -$ нормована експоненційна функція (softmax).

Функція має наступний вигляд:

$$f\ s\ _i = \frac{e^{s_i}}{\sum_j^C e^{s_j}},$$

де C – кількість можливих дій;

$a_t^i = \mathbb{I}(a_t = i)$;

D - набір прикладів роботи експертної стратегії.

Просто кажучи, мережа вчиться вирішувати задачу класифікації спостережень, де класами є дії, мінімізуючи крос-ентропійну помилку.

Якщо ж простір дій є неперервним, то задача стає задачею регресії:

$$\theta = argmin_\theta \sum_{a_t, o_t \in D} h_\theta\ o_t\ - a_t\ _2$$

Тобто параметри знаходяться в результаті мінімізації L_2 помилки.

Алгоритм клонування поведінки можна проілюструвати наступною схемою (рисунок 2.8):

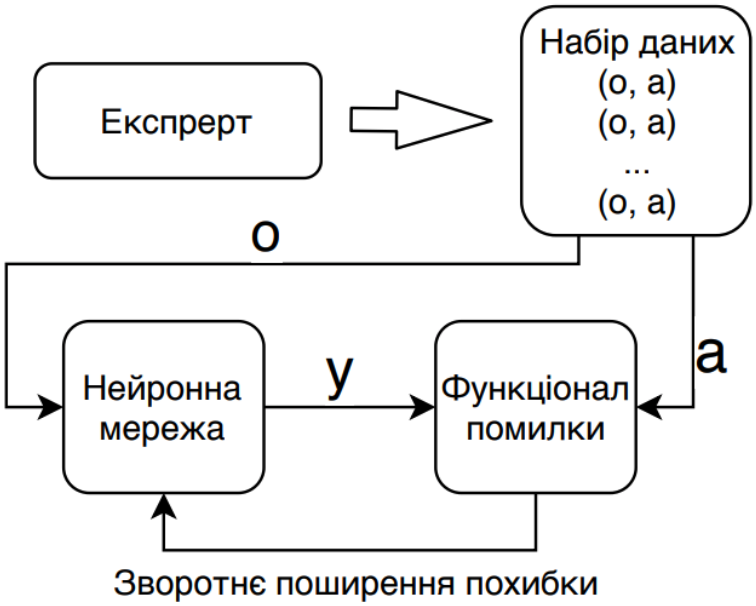


Рисунок 2.8 - Схема роботи алгоритму клонування поведінки

Такий алгоритм має суттєві недоліки. По-перше, функція, що наближує нейронна мережа, не є визначеною для всіх можливих спостережень, тому, якщо під час роботи вивчена стратегія вийде за рамки знайомих траєкторій, його поведінка стане невизначеною і він, імовірно, стане сильніше відхилятися від них, що призведе до збоїв в його роботі. По-друге, на практиці цей підхід часто зустрічає багато проблем із навчанням через можливу мультимодальність розподілу, авторегресійну природу експерта, тощо.

Розглянемо модифікацію цього алгоритму, що призначена дещо зменшити існуючі проблеми клонування поведінки. Ця модифікація носить назву DAgger (Dataset Aggregation), і псевдокод цього алгоритму можна бачити на рисунку 2.9:

Ініціалізувати набір прикладів $\mathcal{D}$

Повторювати:

Навчити $\pi_\theta(a_t|o_t)$ використовуючи набір даних $\mathcal{D}$.

Згенерувати новий набір спостережень $\mathcal{D}_\pi^o = (o_1, \dots, o_M)$, використовуючи поточну стратегію.

Отримати дії використовуючи експертний план, доповнюючи $\mathcal{D}_\pi^o$ діями: $\mathcal{D}_\pi = (o_1, a_1, \dots, o_M, a_M)$

Доповнення набору даних: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

Рисунок 2.9 - Псевдокод алгоритму DAgger

В цьому алгоритмі набір даних щоразу доповнюється спостереженнями із траєкторій, згенерованих поточною стратегією, та діями - відповідями експерта на ці спостереженнями (рисунок 2.10).

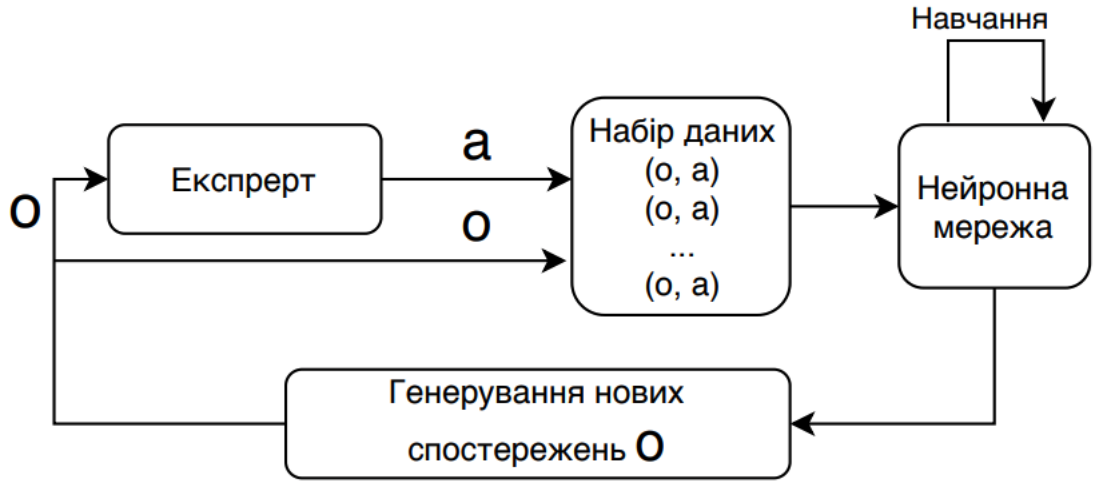


Рисунок 2.10 - Схема роботи алгоритму DAgger

Це дозволяє більш щільно визначати стратегію, проте основною перевагою такого режиму є те, що, на відміну від клонування поведінки, має лінійну залежність відхилу від експертної політики від довжини епізоду замість квадратичної, а тому збігається значно швидше і забезпечує ліпший результат. Теоретичний аналіз цього алгоритму можна знайти в оригінальній статті, що представляє цей алгоритм [35]

2.4 Алгоритми градієнту стратегії

Одним з підходів у навчанні з підкріпленням є диференціювання очікуваної кумулятивної винагороди, використовуючи градієнт стратегії (Policy Gradients).

2.4.1 Виведення градієнту стратегії

Нагадаємо, що функціоналом для максимізації в навчанні з підкріпленням є

$$J\ \theta\ = \mathbb{E}_{\tau\sim\pi_\theta\ \tau}\ \ R\ \tau\ \ ,$$

де $R\ \tau$ – кумулятивна винагорода;

τ – траєкторія;

π_θ – стратегія з параметрами θ.

Запишемо $J(\theta)$ у явному вигляді:

$$J\ \theta\ =\ \ \pi_\theta\ \tau\ R\ \tau\ d\tau$$

Тепер скористаємося наступною рівністю:

$$\pi_\theta \nabla_\theta \log \pi_\theta\ \tau\ = \pi_\theta\ \tau\ \frac{\nabla_\theta \pi_\theta\ \tau}{\pi_\theta(\tau)} = \nabla_\theta \pi_\theta(\tau)$$

Отримуємо:

$$\nabla_\theta J\ \theta\ =\ \ \nabla_\theta \pi_\theta\ \tau\ R\ \tau\ d\tau = \ \ \pi_\theta\ \tau\ \nabla_\theta \log \pi_\theta\ \tau\ R\ \tau\ d\tau =$$

$$= \mathbb{E}_{\tau\sim\pi_\theta\ \tau}\ \ \nabla_\theta \log \pi_\theta\ \tau\ R\ \tau \qquad 1$$

Проте, для обчислення чи наближення градієнту треба знати внутрішню динаміку середовища, оскільки:

$$\pi_\theta\ \tau\ = \pi_\theta\ \ s_1, a_1, \dots, s_T, a_T\ \ = p(s_1) \prod_{t=1}^{T} \pi_\theta\ \ a_t\ s_t\ \ p(s_{t+1}|s_t, a_t)$$

Прологарифмуємо це рівняння, отримаємо:

$$\log \pi_\theta\ \tau\ = \log p\ \ s_1\ \ + \sum_{t=1}^{T} \log \pi_\theta\ \ a_t\ s_t\ \ + \log p\ \ s_{t+1}\ s_t, a_t$$

Візьмемо градієнт обох частин цього рівняння:

$$\nabla_\theta \log \pi_\theta\ \tau\ = \nabla_\theta\ \log p\ s_1\ + \sum_{t=1}^{T} \log \pi_\theta\ a_t\ s_t\ + \log p\ s_{t+1}\ s_t, a_t\ =$$

$$= \nabla_\theta \sum_{t=1}^{T} \log \pi_\theta\ a_t\ s_t\ = \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta\ a_t\ s_t \qquad (2)$$

Підставимо (2) в (1):

$$\nabla_\theta J\ \theta\ = \mathbb{E}_{\tau \sim \pi_\theta\ \tau} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta\ a_t\ s_t \sum_{t=1}^{T} R(s_t, a_t)$$

Математичне сподівання можна наблизити, отримавши незміщену оцінку градієнтів:

$$\nabla_\theta J\ \theta\ \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta\ a_{i,t}\ s_{i,t} \sum_{t=1}^{T} R\ s_{i,t}, a_{i,t} \qquad 3$$

Таким чином, можна записати наступний алгоритм (REINFORCE [36]):

Ініціалізувати довільними значеннями параметри стратегії π_θ

Повторювати:

Генерування набору прикладів, використовуючи стратегію $\pi_\theta(a_t|s_t)$

$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}[(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t))(\sum_{t=1}^{T} R(s_t, a_t))]$

$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Рисунок 2.11 - Псевдокод алгоритму REINFORCE

Насправді, отриманий вираз (3) для оцінки градієнту можна розглядати як узагальнення градієнту з методу максимальної правдоподібності (maximum likelihood):

$$\nabla_\theta J_{ML}\ \theta\ \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\ a_{i,t}\ s_{i,t}$$

Тобто ММП можна отримати з оцінки градієнту стратегії, вважаючи, що всі приклади мають однакову вагу у вигляді винагороди, а сам градієнт стратегії – це ММП, де кожен приклад має різну вагу.

Алгоритм REINFORCE використовує незміщену оцінку градієнта, проте в такому вигляді має досить велику дисперсію, що суттєво сповільнює збіжність на практиці. Однак, цю дисперсію можливо зменшити, про що піде мова в наступному підрозділі.

2.4.2 Зниження дисперсії градієнту

2.4.2.1 Причинність

Основним великої джерелом дисперсії градієнту є розмір винагород, що входять до кумулятивної винагороди у виразі оцінки градієнту стратегії. Тому, для зменшення дисперсії, цей множник потрібно зменшувати.

Першим способом зменшення є так званий принцип причинності. Цей принцип полягає в тому, що стратегія в момент часу t' не може впливати на винагороду в час $t < t'$. Тобто тепер оцінка винагороди стає:

$$\nabla_\theta J\ \theta\ \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\ a_{i,t}\ s_{i,t}\sum_{t\prime=t}^{T} R\ s_{i,t\prime}, a_{i,t\prime}$$

Такий метод на практиці дозволяє суттєво зменшити дисперсію градієнтів, залишаючи оцінку незміщеною [37].

2.4.2.2 Базова вартість

Зменшити дисперсію також можна зменшуючи множник із винагородами, віднімаючи від нього так звану базову вартість що є, наприклад, середньою кумулятивною винагородою:

$$\nabla_\theta J\ \theta\ \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta\ \tau\ \ R\ \tau\ - b\ \ ,$$

$$b = \frac{1}{N}\sum_{i=1}^{N} R\ \tau$$

Якщо b є сталою, то оцінка залишається незміщеною:

$$\mathbb{E}\ \nabla_\theta \log \pi_\theta\ \tau\ b\ = \int \pi_\theta\ \tau\ \nabla_\theta \log \pi_\theta\ \tau\ b\, d\tau\ =$$

$$= \int \nabla_\theta \pi_\theta\ \tau\ \ \mathrm{b}\, d\tau = b\nabla_\theta \int \pi_\theta\ \tau\ d\tau = b\nabla_\theta 1 = 0$$

Середня кумулятивна винагорода не є найкращою базовою вартістю, хоча на практиці працює досить ефективно. Можна показати, що найкращою базовою вартістю є:

$$b = \frac{\mathbb{E}[\ \nabla_\theta \log \pi_\theta\ \tau\ \ ^2 R(\tau)]}{\mathbb{E}[\ \nabla_\theta \log \pi_\theta\ \tau\ \ ^2]}$$

Ця величина являє собою очікуваною винагородою, але зваженою величинами градієнтів. І хоча ця величина і є найкращою базовою вартістю, на практиці вона працює не набагато ефективніше середнього, а також є більш складною для обчислення.

2.5 Алгоритми типу Актор-Критик

2.5.1 Оцінювання переваги

Розглянемо множник із кумулятивної винагородою після застосування принципу причинності:

$$\sum_{t\prime=t}^{T} R\ s_{i,t\prime}, a_{i,t\prime}$$

Цей множник являє собою ніщо інше, ніж оцінку очікуваної нагороди, якщо припустити, що ми вибрали дію $a_{i,t}$, знаходячись в стані $s_{i,t}$. Тобто цей множник можна переписати у вигляді:

$$Q\ s_t, a_t\ = \sum_{t'=t}^{T} \mathbb{E}_{\pi_\theta}[R(s_{t'}, a_{t'})|s_t, a_t]$$

Тепер, розглянемо середню базову вартість для такого випадку:

$$V\ s_t\ = \mathbb{E}_{a_t \sim \pi_\theta\ a_t, s_t}\ [Q\ s_t, a_t\ ]$$

Тепер, вираз для оцінки градієнту стратегія приймає вигляд:

$$\nabla_\theta J\ \theta\ \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta\ a_{i,t}\ s_{i,t}\ A^{\pi_\theta}\ (s_{i,t}, a_{i,t})\ ,$$

$$A^{\pi_\theta}\ s_{i,t}, a_{i,t}\ = Q\ s_{i,t}, a_{i,t}\ - V\ s_{i,t}\ ,$$

де величина A^{π_θ} носить назву переваги (advantage).

Така оцінка має нижчу дисперсію, і чим краща оцінка переваги, тим менша дисперсія. Розглянемо два методи її оцінки.

Для оцінки переваги, можна оцінювати A^{π}, Q^{π} або V^{π}. Методи, які ми розглянемо оцінюють останню функцію. Оцінивши V^{π}, отримаємо вираз для переваги:

$$A^{\pi}\ s_t, a_t\ \approx R\ s_t, a_t\ + V^{\pi}\ s_{t+1}\ - V^{\pi}(s_t)$$

Сама V^{π} в глибокому навчанні з підкріпленням може наближатись окремою нейронною мережею, чи тією ж самою мережею, що наближує стратегію. Проте, в тому, як саме навчати таку мережу, і полягає різниця між двома наступними методами:

1. Оцінка Монте-Карло

$$V^{\pi}\ s_t\ \approx \sum_{t'=t}^{T} R\ s_{t'}, a_{t'}$$

Ця оцінка має значно більшу дисперсію, ніж

$$V^{\pi}\ s_t\ \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t'=t}^{T} R\ s_{t'}, a_{t'}$$

Проте останню оцінку отримати неможливо, адже вона потребує багатократного повторення епізоду з моменту t, що не є можливим у більшості реальних застосувань. Таким чином, ми дещо збільшуємо дисперсію оцінки, однак, на практиці, навіть така оцінка працює достатньо ефективно. В такому разі, приклади, на яких вчиться мережа, що наближує V^{π} виглядають так:

$$y_{i,t} = \sum_{t'=t}^{T} R\ s_{i,t'}, a_{i,t'}\ ,$$

$$s_{i,t}, y_{i,t}$$

А функціонал похибки:

$$L\ \theta\ = \frac{1}{2} \sum_i V_\theta^\pi\ s_i\ - y_i\ ^2$$

2. Бутстрап оцінка (bootstapped estimate)

$$V^\pi\ s_t\ \approx R\ s_{i,t}, a_{i,t}\ + V_\theta^\pi\ s_{i,t+1}$$

Тут значення функції вартості оцінюється використовуючи поточну її оцінку. Така оцінка може бути дещо нестабільною, хоча на практиці вона також є досить ефективною. В такому разі, приклади, на яких вчиться мережа, що наближує функцію вартості мають вигляд:

$$y_{i,t} = R\ s_{i,t}, a_{i,t}\ + V_\theta^\pi\ s_{i,t+1}\ ,$$
$$s_{i,t}, y_{i,t}$$

А функціонал похибки:

$$L\ \theta\ = \frac{1}{2} \sum_i V_\theta^\pi\ s_i\ - y_i\ ^2$$

Зауважимо, що в обох методах, цільові значення y_i використовуються, як константи. Тобто градієнт не проходить далі в їх складові при зворотному поширенні похибки.

2.5.2 Алгоритм Актор-Критик

Тепер, використовуючи один з методів наближення отримуємо алгоритм, що носить назву Актор-Критик (рисунок 2.12):

Повторювати:

Згенерувати траєкторії $\{(s_i, a_i)\}$, використовуючи $\pi_\theta(a|s)$

Оновити ваги $\hat{V}_\theta^\pi(s)$ (1 методом)

$\hat{A}^\pi(s_i, a_i) = R(s_i, a_i) + \hat{V}^\pi(s_{i+1}) - \hat{V}^\pi(s_i)$

$\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(a_i|s_i)\hat{A}^\pi(s_i, a_i)$

$\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

Рисунок 2.12 - Псевдокод алгоритму Актор-Критик

Згадаємо, що кумулятивна винагорода може бути дисконтованою. Це дозволяє зважити важливість прикладів, забезпечити швидшу збіжність суми, а також дещо зменшити дисперсію. Наведемо дві версії алгоритму Актор-Критик із дисконтуванням: онлайн (рисунок 2.13) та батч версію (рисунок 2.14):

Повторювати:

Вибрати дію $a \sim \pi_\theta(a|s)$, отримати (s, a, s', R)

Оновити ваги $\hat{V}_\phi^\pi(s)$ (2 методом)

$\hat{A}^\pi(s_i, a_i) = R(s, a) + \gamma\hat{V}^\pi(s') - \hat{V}^\pi(s)$

$\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(a|s)\hat{A}^\pi(s, a)$

$\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

Рисунок 2.13 - Псевдокод онлайн версії алгоритму Актор-Критик з дисконтуванням

Повторювати:

Згенерувати траєкторії $\{(s_i, a_i)\}$, використовуючи $\pi_\theta(a|s)$

Оновити ваги $\hat{V}_\theta^\pi(s)$ (1 методом)

$\hat{A}^\pi(s_i, a_i) = R(s_i, a_i) + \gamma\hat{V}^\pi(s_{i+1}) - \hat{V}^\pi(s_i)$

$\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(a_i|s_i)\hat{A}^\pi(s_i, a_i)$

$\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

Рисунок 2.14 - Псевдокод батч версії алгоритму Актор-Критик з дисконтуванням

Назва Актор-Критик походить від пари моделей, які наближують функцію вартості та стратегію. Мережа, що наближує стратегію зветься Актором, а мережа (чи її частина у випадку розділення вагів), яка наближує функцію вартості називається Критиком.

2.6 Алгоритм Наближеної оптимізації стратегії

2.6.1 Проблема повторного використання прикладів

Одним із великих недоліків наведених алгоритмів Актор-Критик є факт того, що після кожного оновлення стратегії треба знову генерувати приклади. Це досить сильно сповільнює збіжність та швидкість ітерацій. В цьому підрозділі розглянемо прийом, який дозволяє в деякій мірі вирішити цю проблему. Цей метод носить назву IS (Importance Sampling). Він дозволяє визначати математичне сподівання за новим розподілом, використовуючи старий розподіл, та знаючи їх відношення:

$$\mathbb{E}_{x\sim p_{new}\ x}\ f\ x\ = \ p_{new}\ x\ f\ x\ dx = \ \frac{p_{old}\ x}{p_{old}\ x} p_{new}\ x\ f\ x\ dx =$$

$$= \ p_{old}\ x\ \frac{p_{new}\ x}{p_{old}\ x} f\ x\ dx = \mathbb{E}_{x\sim p_{old}\ x}\ \frac{p_{new}\ x}{p_{old}\ x} f(x)$$

Використовуючи це відношення спробуємо вивести вираз для оцінки градієнту стратегії з використанням старої стратегії для генерування прикладів.

$$\pi_\theta\ \tau\ = p(s_1) \prod_{t=1}^{T} \pi_\theta\ a_t\ s_t\ p\ s_{t+1}\ s_t, a_t\ ,$$

$$\frac{\pi_\theta\ \tau}{\pi'_\theta\ \tau} = \frac{p\ s_1 \prod_{t=1}^{T} \pi_\theta\ a_t\ s_t\ p\ s_{t+1}\ s_t, a_t}{p\ s_1 \prod_{t=1}^{T} \pi'_\theta\ a_t\ s_t\ p\ s_{t+1}\ s_t, a_t} = \frac{\prod_{t=1}^{T} \pi_\theta\ a_t\ s_t}{\prod_{t=1}^{T} \pi'_\theta\ a_t\ s_t},$$

$$\nabla_{\theta'} J\ \theta'\ = \ \mathbb{E}_{\tau\sim\pi_\theta\ \tau}\ \frac{\pi_{\theta'}\ \tau}{\pi_\theta\ \tau} \nabla_{\theta'} \log \pi_{\theta'}\ \tau\ R\ \tau\ =$$

$$= \mathbb{E}_{\tau\sim\pi_\theta\ \tau}\ \frac{\prod_{t=1}^{T} \pi_\theta\ a_t\ s_t}{\prod_{t=1}^{T} \pi'_\theta\ a_t\ s_t} \sum_{t=1}^{T} \nabla_{\theta'} \log \pi_{\theta'}\ a_t|s_t \sum_{t=1}^{T} R(s_t, a_t\ =$$

$$= \text{причинність} =$$

$$= \mathbb{E}_{\tau\sim\pi_\theta\ \tau} \sum_{t=1}^{T} \nabla_{\theta'} \log \pi_{\theta'}\ a_t|s_t \sum_{t'=1}^{t} \frac{\pi_\theta\ a_{t'}\ s_{t'}}{\pi'_\theta\ a_{t'}\ s_{t'}} \times$$

$$\times \sum_{t'=t}^{T} R(s_{t'}, a_{t'}) \sum_{t''=t}^{t'} \frac{\pi_\theta\ a_{t''}\ s_{t''}}{\pi'_\theta\ a_{t''}\ s_{t''}} =$$

$$= \mathbb{E}_{\tau\sim\pi_\theta\ \tau} \sum_{t=1}^{T} \nabla_{\theta'} \log \pi_{\theta'}\ a_t|s_t \sum_{t'=1}^{t} \frac{\pi_\theta\ a_{t'}\ s_{t'}}{\pi'_\theta\ a_{t'}\ s_{t'}} \sum_{t'=t}^{T} R(s_{t'}, a_{t'})$$

Член з добутком сильно збільшує дисперсію оцінки, адже є експоненційним за t. Дещо перепишемо цільовий функціонал:

$$\theta^* = argmax_\theta \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim p_\theta\ s_t,a_t}\ [R\ s_t, a_t\ ]$$

Тобто тепер він складається із суми математичних сподівань за маргінальними станами і діями.

$$J\ \theta\ = \sum_{t=1}^{T} \mathbb{E}_{\ s_t,a_t\ \sim p_\theta\ a_t,s_t}\ R\ s_t, a_t\ = \sum_{t=1}^{T} \mathbb{E}_{s_t\sim p_\theta\ s_t}\ \mathbb{E}_{a_t\sim\pi_\theta\ a_t|s_t}\ R\ s_t, a_t\ ,$$

$$J\ \theta' = \sum_{t=1}^{T} \mathbb{E}_{s_t\sim p_\theta\ s_t}\ \frac{p_{\theta'}(s_t)}{p_\theta\ s_t} \mathbb{E}_{a_t\sim\pi_\theta\ a_t|s_t}\ \frac{\pi_{\theta'}\ a_t\ s_t}{\pi_\theta\ a_t\ s_t} R\ s_t, a_t \approx$$

$$\approx \sum_{t=1}^{T} \mathbb{E}_{s_t\sim p_\theta\ s_t}\ \mathbb{E}_{a_t\sim\pi_\theta\ a_t|s_t}\ \frac{\pi_{\theta'}\ a_t\ s_t}{\pi_\theta\ a_t\ s_t} R\ s_t, a_t$$

2.6.2 Наближена оптимізація стратегії

Алгоритм наближеної оптимізації стратегії (Proximal Policy Optimization, далі PPO) використовує дещо модифікований підхід із минулого підрозділу. Основною ідеєю є те, що під час оновлень стратегії за градієнтами, оціненими з використанням IS (Importance Sampling), не можна допускати ситуацію, коли поточна стратегія та стара стратегія досить сильно відрізняються. Більш детально про цей алгоритм можна знайти в оригінальній статті, що представляє цей алгоритм [24]. Цей алгоритм вирішує задачу:

$$\mathbb{E}_{\ s_t,a_t\ \sim p_\theta(s_t,a_t)}\ \frac{\pi_{\theta'}\ a_t\ s_t}{\pi_\theta\ a_t\ s_t} A^{\pi_\theta}\ s_t,a_t \quad \rightarrow max$$

При обмеженні:

$$D_{KL}\ \pi_{\theta'}\ a_t\ s_t \quad \pi_\theta\ a_t\ s_t \quad \leq \epsilon,$$

де $D_{KL}(\quad)$ – дивергенція Кульбака-Лейблера, що є мірою віддаленості двох розподілів;

ϵ- деяка константа.

Означення КЛ-дивергенції:

$$D_{KL}\ p(x)\ \ q(x)\ = -\ \ \ \ p\ x\ \log\frac{p\ x}{q\ x},$$

де p та q – деякі розподіли.

Обмеження можна злити із цільовим функціоналом:

$$\mathbb{E}_{\ s_t,a_t\ \sim p_\theta\ s_t,a_t}\ \ \frac{\pi_{\theta'}\ a_t\ s_t}{\pi_\theta\ a_t\ s_t} A^{\pi_\theta}\ s_t,a_t \quad -$$

$$-\mathbb{E}_{\ s_t,a_t\ \sim p_\theta\ s_t,a_t}\ \ \beta D_{KL}(\pi'_\theta\ \cdot\ s_t\ \ ||\ \pi_\theta\ \cdot\ s_t\ ) \ \rightarrow max,$$

де β - гіперпараметр, що контролює вагу штрафу

В цьому випадку обмеження розглядається як штрафна функція.

В рамках алгоритму вибір ваги штрафу відбувається динамічним чином: якщо штраф менше деякого порогу, який також є гіперпараметром, то β збільшується, а якщо більше – зменшується.

Іншим підходом є PPO з обрізкою. Позначимо $r_t\ \theta\ = \frac{\pi_{\theta\prime}(a_t|s_t)}{\pi_\theta(a_t|s_t)}$. В рамках цього підходу, відношення розподілів $r_t(\theta)$ тримається в діапазоні $1-\epsilon, 1+\epsilon$. Цільовий функціонал в такому випадку приймає вигляд:

$$\mathbb{E}_{\ s_t,a_t\ \sim p_\theta\ s_t,a_t}\ \min\ r_t\ \theta\ A^{\pi_\theta}\ s_t,a_t\ ,clip\ r_t\ \theta\ ,1-\epsilon,1+\epsilon\ A^{\pi_\theta}\ s_t,a_t\ \ ,$$

де $clip\ x,a,b\ = \min(\max\ x,a\ ,b)$.

Такий функціонал штрафує за великі зміни в стратегії, якщо він знаходиться в частині траєкторій, де він отримує невелику винагороду. Цей підхід використовується більш часто. Він має всього один додатковий гіперпараметр для підбору. Псевдокод, цього алгоритму можна бачити на рисунку 2.15:

Ініціалізувати стратегію параметрами θ_0, вибрати поріг ϵ

Для $k = 0,1,\ldots$ виконати:

Згенерувати набір прикладів, використовуючи стратегію $\pi_{\theta_k}(a_t|s_t)$

Оцінити переваги A^{π_k}, використовуючи один з методів його оцінки (Критик)

Оновити стратегію:

$$\theta_{k+1} = argmax_\theta\ J_{\theta_k}^{CLIP}(\theta)$$

Здійснюючі K кроків градієнтного спуску (Adam), де

$$J_{\theta_k}^{CLIP}(\theta) = \mathbb{E}_{\tau\sim\pi_k}\sum_t\ \left[\min\left(r_t(\theta)A^{\pi_\theta}(s_t,a_t), clip(r_t(\theta),1-\epsilon,1+\epsilon)A^{\pi_\theta}(s_t,a_t)\right)\right]$$

Рисунок 2.15 - Псевдокод алгоритму PPO

2.7 Методи вирішення проблеми дослідження

2.7.1 Дилема дослідження та експлуатування

Однією із найбільш фундаментальних проблем навчання з підкріпленням є, так звана, дилема дослідження та експлуатації. Дослідженням називають здатність алгоритму знаходити нові стратегії, що можуть виявитися більш оптимальними. Проте, можливо і не виявляться. В такому разі, бажаною поведінкою була б деяка збіжність алгоритму, підкріплена тим, що кумулятивна винагорода зменшується при змінах в стратегії. Проблема дослідження полягає в тому, що алгоритм важко змусити не переставати активно шукати нові стратегії, особливо з плином часу. Крім того, коли винагорода є досить розрідженою, тобто для отримання інформації, що може підтвердити правильність дій агенту, треба зробити довгу серію структурованих дій. Але на початку роботи, алгоритм є дуже шумним, проявляючи ознаки гаусівського процесу. Тому, в таких випадках, проходить дуже велика кількість часу, поки алгоритм випадково знайде цю одиницю винагороди і почне направлену оптимізацію стратегії. Прикладом таких середовищ можна назвати, наприклад, відому Montezuma’s Revenge (рисунок 2.16):

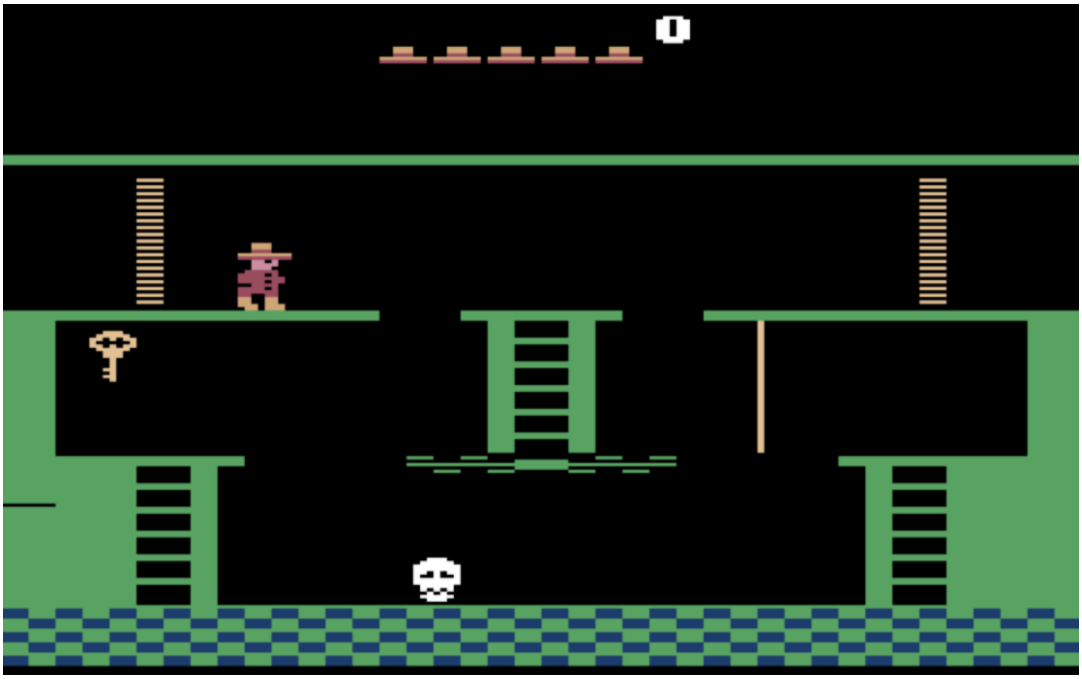


Рисунок 2.16 - Зображення із гри Montezuma’s Revenge

В цій грі агенту потрібно пройти велику кількість кімнат для проходження рівня. По всьому рівню розкидані двері, ключі від них та небезпечні об’єкти, а

винагорода є дуже розрідженою. Агенту дуже важко пройти першу кімнату, де треба взяти ключ і відкрити ним двері, попри відсутність нагороди.

Для вирішення таких проблем в глибокому навчанні існує декілька методів, проте вони є активною сферою для досліджень.

2.7.2 Метод підрахунків

Одним із способів заохотити агент до досліджування є додаткова нагорода за перехід в стани, в яких він ще не був, або був досить рідко. Ця ідея лягла в основу класичного методу підрахунків [40].

В рамках цього метода до винагороди, яку агент щоразу отримує, додається додатковий бонус у вигляді:

$$\mathrm{R}_{\mathrm{count}}\ s, a\ = \frac{\beta}{\overline{n\ s, a}},$$

де $\mathrm{n}(\mathrm{s}, \mathrm{a})$ – кількість разів, коли агент був в стані s із дією a;

β – підлаштовуємий параметр.

Відношення із зворотнім квадратним коренем є оптимальним для дослідження, що показано в [38].

На жаль, цей метод перестає працювати, коли розмірності просторів станів та дій збільшуються, адже, по-перше, для пам’ять зберігання $\mathrm{n}(\cdot,\cdot)$ оцінюється, як $\mathrm{O}\ \ S \cdot A\ $. Крім того, сам метод суттєво сповільнюється, адже при великій розмірності цих просторів, повторне відвідування однієї пари стану-дії стає значно менш імовірним.

На практиці, коли розмірності дуже великі, підмножини близьких пар станів-дій є дуже схожими, а тому такі відвідування таких підмножин можна підраховувати і використовувати їх як стани в методі підрахунків. Для цього залишається лише знайти хеш-функцію $\phi: S \to \mathbb{Z}$. В цьому випадку, бонус записується як:

$$\mathrm{R}_{\mathrm{count}}\ s\ = \frac{\beta}{n\ \phi(s)}$$

Хеш-функцію використовувати можна, як вигадану самостійно, яка включає апріорне знання про середовище, або вивчити окремо під час навчання. Більш детальну інформацію про другий варіант, як і загалом про цей метод можна знайти в оригінальній статті #Explore [39].

2.7.3 Навчання засноване на допитливості

Іншим методом вирішення проблеми дослідження є так зване навчання засноване на допитливості (Curiosity-driven exploration). Основною ідеєю є навчання додаткової моделі, так званого модуля внутрішньої допитливості (Intrinsic Curiosity Module). Ця модель складається з трьох частин: прямої моделі, оберненої моделі та екстрактору ознак (рисунок 2.17):

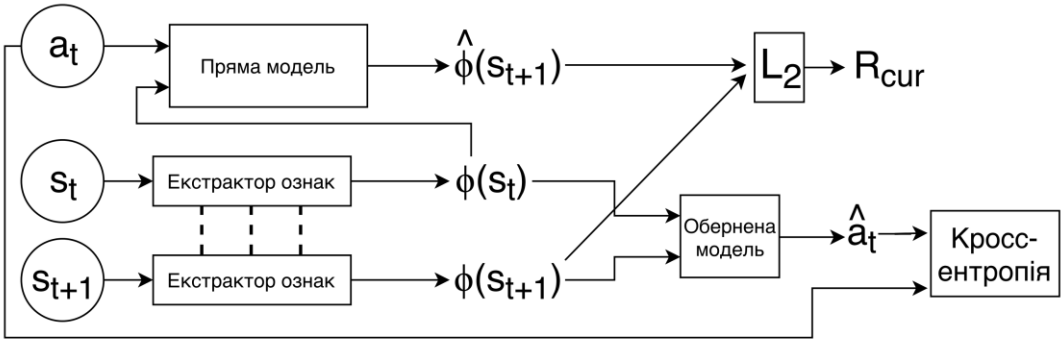


Рисунок 2.17 - Схема роботи навчання модулю внутрішньої допитливості

Екстрактор ознак зменшує розмірність самого стану, представляючи собою відображення $\phi: S \to \mathbb{R}^m$. Пряма модель - $ICM_f: A \times \mathbb{R}^m \to \mathbb{R}^m$, є регресією, яка намагається прогнозувати вектор ознак наступного стану за вектором ознак поточного стану і дії, яку агент вибрав. Середньоквадратичне відхилення прогнозу вектору ознак наступного стану і дійсного вектору ознак, що використовується як функціонал помилки при навчанні, також використовується як бонусна винагорода R_{cur}. Це відхилення являє собою міру несподіваності ICM,

при спостереженні нового стану. Тобто великий бонус отримує агент, коли досліджує несподівані та нові траєкторії.

Проте, якщо навчати ICM тільки із прямою моделлю, екстрактор ознак може почати підлаштовуватись під регресію і вивчати схожі представлення для всіх станів. Для того, щоб такого не відбувалося, існує обернена модель, яка за поточним та наступним векторами ознак відновлює дію, яку агент здійснив. Більш детальне обґрунтування методу можна знайти в оригінальній статті [41].

Цікаво зауважити, що навіть без основного сигналу підкріплення, модель, що використовує ICM досить успішно вивчає корисні стратегії у незнайомих середовищах. А крім того, на практиці результат навчання покращується, якщо спочатку навчати модель тільки з винагородою за допитливість, а потім підключати основний сигнал підкріплення.

2.8 Використання представлень середнього рівня

Алгоритми глибокого навчання з підкріпленням потребують великої кількості часу для збіжності. Це пояснюється не тільки часом, затраченим на пошук оптимальної стратегії, а ще й часом, затраченим на навчання екстрактору ознак, задачею якого є вивчення відображення із простору станів у лінійний простір векторів меншої розмірності.

В глибокому навчанні велику роль грає ефект, що називається Transfer Learning, тобто перевикористання вивчених мереж для нових задач. В рамках цього підходу мережі розглядаються як пара кодер-декодер (encoder-decoder). Кодером називають саме екстрактор ознак, які потім використовуються декодером для подальшого вирішення поставленої задачі. Наприклад, у класифікаторів декодер може являти собою один або декілька останніх шарів мережі, де розмірність виходу останнього шару співпадає з кількістю класів. Після навчання з нуля такої мережі можна взяти вже навчений кодер та повторно використовувати його для іншої задачі.

Автори статті про використання представлень середнього рівня [42] пропонують ідею, схожу на [43]. Вони використовують кодери із мереж, навчених для різних задач: класифікації, детекції, сегментації та інших (рисунок 2.18).

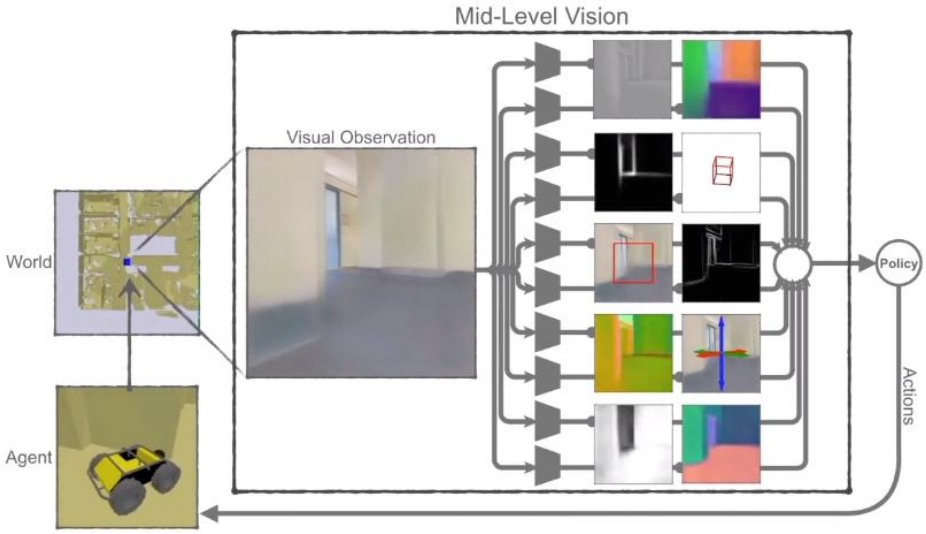


Рисунок 2.18 - Схема, що ілюструє ідею використання ознак середнього рівня

У статті автори надають порівняльний аналіз використання різних кодерів та їх комбінацій для навчання агентів для навігації (рисунок 2.19).

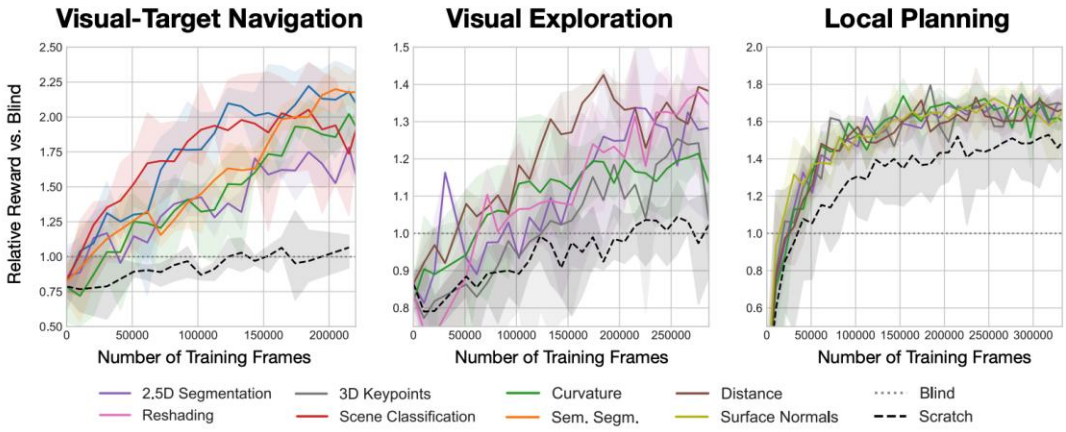


Рисунок 2.19 - Графіки середньої винагороди для різних кодерів

2.9 Висновки

В цьому розділі був приведений детальний аналіз методів глибокого навчання з підкріпленням. На початку розділу була наведена таксономія і класифікація існуючих алгоритмів. Особлива увага приділяється алгоритмам оптимізації стратегії.

В рамках аналізу алгоритмів оптимізації стратегії була наведена теорема про градієнт стратегії, яка лежить в основі цього класу алгоритмів. Першим алгоритмом, що наївно використовує цю теорему є REINFORCE. Він дає незміщену оцінку градієнту стратегії, однак має досить велику дисперсію. Далі в цьому розділі розглядаються методи зменшення дисперсії, такі як принцип причинності та поняття базової вартості.

Модифікацією цього алгоритму є Актор-Критик, в якому вводиться поняття переваги, яка оцінюється з використанням інших моделі, що називається Критиком. Критик навчається відновлювати функцію вартості, за допомогою якої будується оцінка переваги. Навчання критика відбувається різними методами, два з яких наведені в цій роботі: Монте-Карло і бутстрап.

Одним із великих недоліків наведених версій алгоритмів Актор-Критик є те, що для кожного оновлення стратегії потребується щоразу генерувати нові приклади, використовуючи поточну версію стратегії. Для часткового вирішення цієї проблеми використовується прийом, що має назву Importance Sampling. Цей прийом дозволяє обчислювати математичне сподівання функції від випадкової величини використовуючи розподіл, що відрізняється від розподілу цієї величини.

Алгоритм наближеної оптимізації стратегії (PPO) використовує IS для уникнення необхідності генерування нових прикладів після кожного оновлення стратегії. Ідея PPO полягає в тому, що оновлена стратегія не сильно відрізняється від своєї попередньої версії, особливо коли кумулятивна винагорода є досить малою. Для цього існує два підходи: штраф дивергенцією Кульбака-Лейблера та обрізання відношення старого і нового розподілів дій, отриманих із стратегії.

Далі розглядається фундаментальна проблема навчання з підкріпленням, що має назву дилеми дослідження та експлуатації. Проблема з дослідженням гостро стоїть перед агентами в середовищах с розрідженою винагородою, в яких агентам проблематично отримати першу винагороду, що б дозволило оптимізувати стратегію. Розглядаються два метода покращення дослідження. Перший пропонує підраховувати кількість відвідувань різних станів і видавати бонусну винагороду за відвідування рідких станів. Другий пропонує навчати додаткову модель, яка намагається прогнозувати ознаки наступного стану. Помилка цієї моделі зростає, коли агент досліджує несподівані та нові траєкторії.

В останньому підрозділі розглядається метод прискорення навчання агентів глибокого навчання з підкріпленням, що отримують візуальні спостереження. Метод повторно використовує ваги кодерів із мереж для задач комп’ютерного зору.

Методи, які описуються в цьому розділі використовуються в цій роботі для вирішення задач навігації, про що піде мова в наступному розділі.

РОЗДІЛ 3 ВИРІШЕННЯ ЗАДАЧІ ВІЗУАЛЬНОЇ НАВІГАЦІЇ МЕТОДАМИ ГЛИБОКОГО НАВЧАННЯ З ПІДКРІПЛЕННЯМ

3.1 Симулятор та середовище

3.1.1 Постановка задачі

Задача, що вирішувалась в цій роботі – задача візуальної навігації. Заданий набір з великої кількості моделей інтер'єрів квартир та домів, деякі з яких можуть мати декілька поверхів. Кожна модель називається сценою. Для кожної сцени існує набір епізодів. Епізод являє собою початкову точку на сцені та кінцеву.

Простір спостережень - кольорове зображення 256 на 256, карта глибин 256 на 256, кут та відстань до цілі.

Простір дій є дискретним і складається із 4 можливих варіантів: рух вперед на 0.25 метра, повороти вправо і вліво на 10 градусів, дія «стоп» яка завершує епізод.

Метрикою була вибрана SPL. Детальніше про неї і функцію винагороди можна знайти в першому розділі.

3.1.2 Симулятор та дані

Для візуалізації та рендерінгу в даній роботі використовувався двигун HabitatSim [44], розроблений Facebook Research саме для таких цілей. Серед аналогів також є Gibson, що розроблюється Stanford Research, проте через більш пристосований API на мові Python, а також факт того, що саме цей двигун використовувався в конкурсі Habitat Challenge, в рамках якого проводилася валідація результатів. Двигун дозволяє візуалізовувати набори із сцен, які являють собою файли із полігональними сітками.

В якості набору сцен був вибраний Gibson 3D (рисунок 3.1). Перевага була віддана цьому набору, оскільки на відміну від SUNCG він складається із відсканованих реальних 3-вимірних приміщень. Знову ж таки, цей набір даних

використовувася в конкурсі Habitat Challenge, тому що на відміну від Matterport3D, частина сцен із Gibson не є публічно доступними, що давало можливість сформувати тестовий набір.

Рисунок 3.1 - Зверху – 3-вимірні моделі сцени із набору даних Gibson. Знизу – панорама із цієї сцени

Крім того, цей набір був розділений на 3 підмножини: тренування, валідацію та міні-валідацію.

3.2 Запропонована модель для навчання

Для роботи був вибраний алгоритм PPO із обрізкою.

В якості мережі, що наближує стратегію була вибрана модель, що складається з трьох частин. Перша - згортковий екстрактор ознак із карт глибини. Схема зображена на рисунку 3.2:

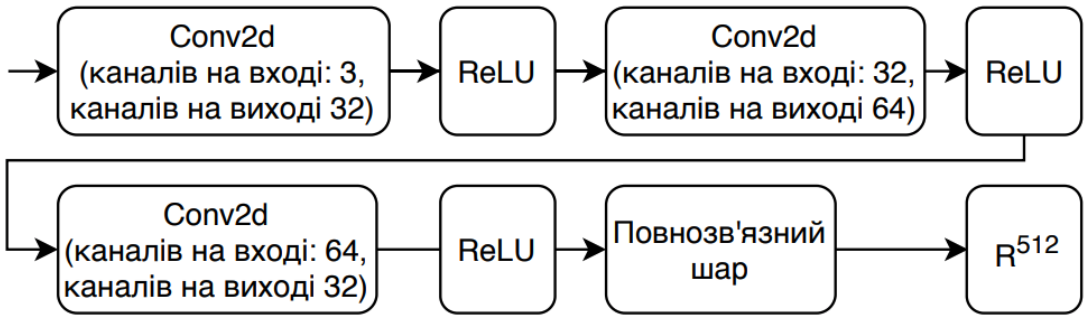


Рисунок 3.2 - Архітектура згорткового екстрактора ознак

Друга - кодер із неглибокого (ResNet18) класифікатора сцен, натренованого на наборі даних Places365 [45], взятого із публічно доступних моделей MIT CSAIL. Зазначимо, що його ваги не оновлюються в процесі навчання, тобто є замороженими. А третя – GRU-ядро (Gated Recurrent Unit), яке і являє собою стратегію. Структура моделі зображена на наступній схемі (рисунок 3.3):

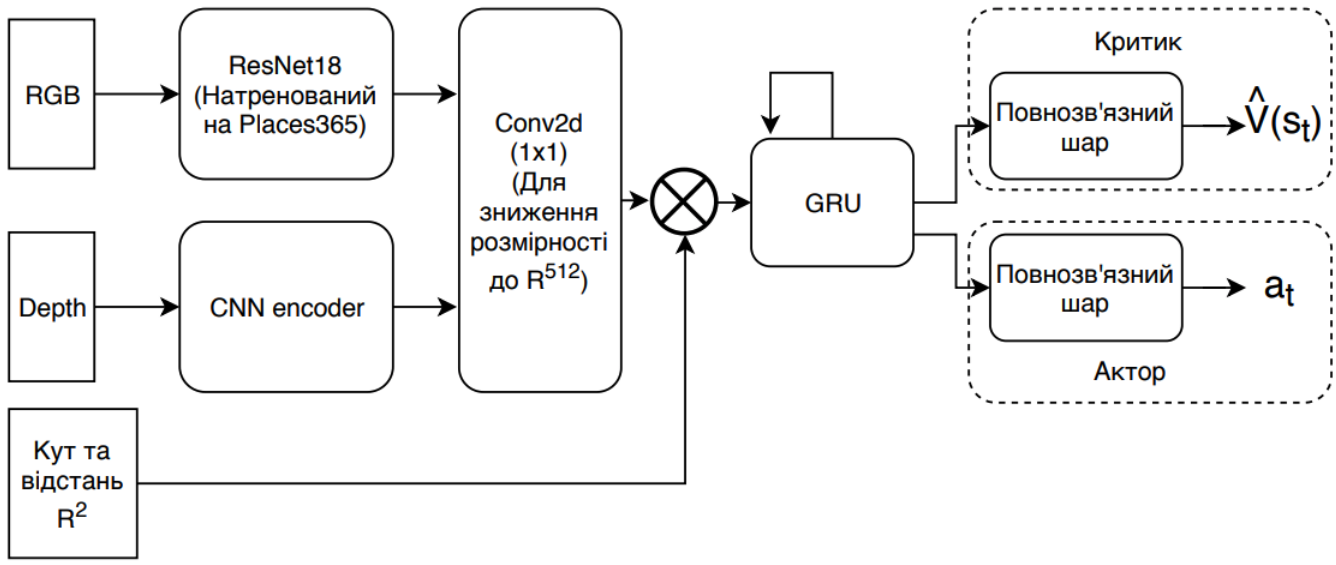


Рисунок 3.3 - Архітектура мережі, що наближує стратегію

3.3 Використані методи вирішення проблеми дослідження

В рішенні були використані два методи заохочення до агенту до дослідження. Перший метод – метод підрахунків, включає в себе необхідність створення хеш-функції $\phi: S \to \mathbb{Z}$. У випадку візуальної навігації, досить природнім вибором цієї функції є функція, яка дискретизує приміщення на квадратні області. Проте, в такому вигляді це працювати не буде. Епізодів на сцені досить багато, а тому навіть початкові точки всіх епізодів досить щільно покривають площу приміщення, а тому, якщо підрахунок робити для всіх епізодів кожної сцени одночасно, значення $n(\cdot)$ швидко стануть дуже великими і бонус зникне. Тому, цей метод потребує модифікації. Замість

$$n\ i\ = n\ i\ + 1$$

Коли

$$\phi\ s\ = i$$

Використовуємо:

$$n\ i\ = n\ i\ + \alpha d_{geo}\ s, s_0\ d_{geo}\ s_0, s_{last}\ ,$$

де d_{geo} – геодезична відстань;

s_0 – початковий стан;

s_{last} – кінцевий стан;

α – гіперпараметр.

Таким чином, заохочується пошук більш віддалених від початкового станів, особливо коли відстань до кінцевої точки є великою. Ця функція прискорює дослідження, особливо в складних епізодах, коли відстань є великою.

Другий метод – метод навчання, заснованого на допитливості. Тут екстрактор ознак має архітектуру, аналогічну екстрактору ознак із стратегії.

Пряма і обернена моделі складаються із двох повнозв’язних шарів із 64 та 32 нейронів.

3.4 Структура реалізації програми для моделювання

Для реалізації була вибрана мова Python 3.6, адже в сучасності вона однозначно домінує в машинному навчанні. Її значна популярність зумовлена зручністю для пототипування та великою кількістю розвинених та якісних бібліотек та фреймворків для машинного навчання та інфраструктури навколо нього.

В якості фреймворку для глибокого навчання був вибраним PyTorch. Цей фреймворк заснований на динамічних обчислювальних графах, та є одним з двох найпопулярніших рішень на сьогодні. Він також дозволяє використовувати відеокарти (далі GPU, Gaphical Processing Unit) для обчислень, що суттєво прискорює навчання. Для цієї роботи використовувались 4 GPU RTX2080Ti, хоча один процес навчання займав тільки 2.

Симулятор використовує GPU для обчислень, а тому кількість процесів, в яких можуть проводитись обчислювання, обмежена пам’яттю самої GPU. Таким чином із двох відеокарт, лише одна використовувалась для навчання, а інша – для обчислень симулятора.

Симулятор надає інтерфейс, схожий на інтерфейс OpenAI Gym, а також надає можливість виконання одразу декількох процесів симулятора одночасно.

Структура програми зображена на наступній схемі (рисунок 3.4):

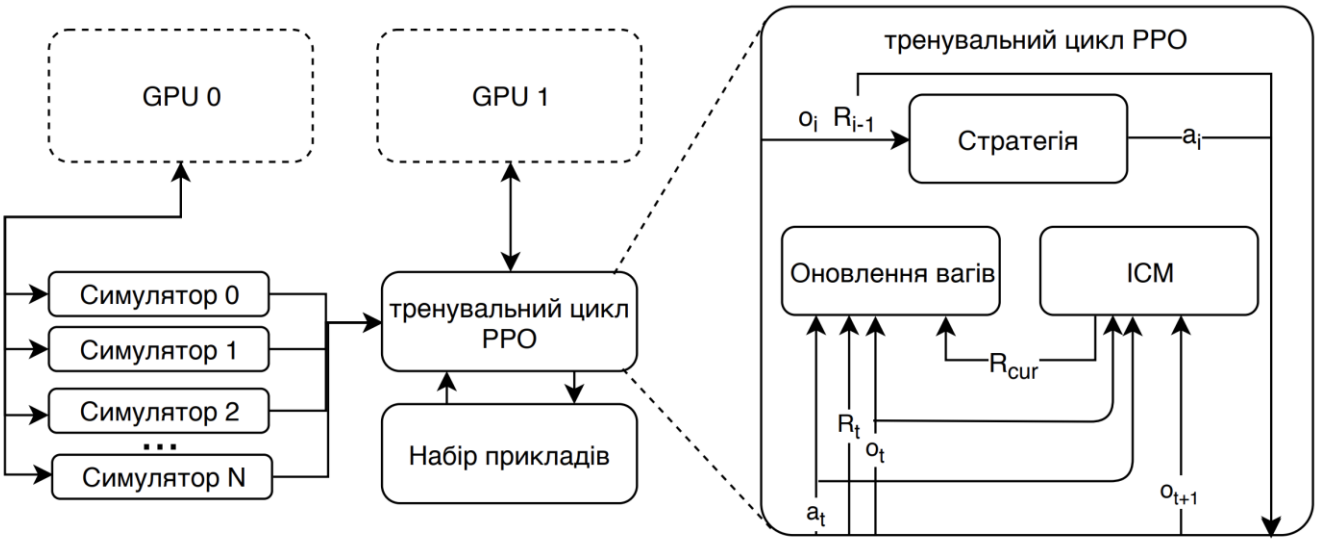


Рисунок 3.4 - Структура програмного рішення

Слід зауважити, що перед основним навчанням було проведене імітаційне навчання алгоритмом DAgger. В якості експерта використовувався алгоритм, що слідує геодезичним шляхом.

3.5 Висновки

В цьому розділі було розглянуто рішення, запропоноване для задачі візуальної навігації. Детально описана задача, симулятор та набори даних, що використовувались для навчання.

Далі наводиться детальна структура нейронної мережі, яка використовувалася для наближення стратегії. Вона складається з трьох частин: двох екстракторів ознак та GRU-ядра. Описується спосіб застосування методів заохочення дослідження: для методу підрахунків наводиться його модифікація та хеш-функція, а для ICM наводиться структура її частин.

Описуються мова та технології, що використовувалися при програмуванні моделі та інфраструктури для неї, архітектура програмного рішення, а також схема його структури.

РОЗДІЛ 4 РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ ТА ЇХ АНАЛІЗ

4.1 Криві навчання

В результаті навчання, найкраща модель досягає значення метрики SPL=0.944 на валідаційному наборі епізодів. Серед них є епізоди на сценах, яких немає в тренувальному наборі, тому такий показник характеризує узагальнюючу здібність отриманої моделі, тобто вона показує хороші результати на сценах, яких не «бачила».

Було проведено 4 експериментів, в рамках яких використовувались різні набори методів:

- Алгоритм PPO
- Алгоритм PPO з використанням попереднього тренування імітаційним моделюванням
- Алгоритм PPO з використанням попереднього тренування імітаційним моделюванням та методом підрахунків.
- Алгоритм PPO з використанням попереднього тренування імітаційним моделюванням, методом підрахунків та методом навчання, заснованого на допитливості.

Графік метрики SPL на валідаційному наборі епізодів протягом навчання можна бачити на рисунку 4.1:

Рисунок 4.1 - Графік метрики SPL на валідаційній вибірці

Для четвертої, найкращої моделі нижче показані графіки основних показників під час навчання: винагороди, підрахованої за вікном довжини 50, середньої, мінімальної та максимальної винагороди, ентропії отриманої стратегії та оцінки дивергенції Кульбака-Лейблера між сусідніми версіями стратегії, що вимірює середній розмір змін під час навчання (рисунок 4.2):

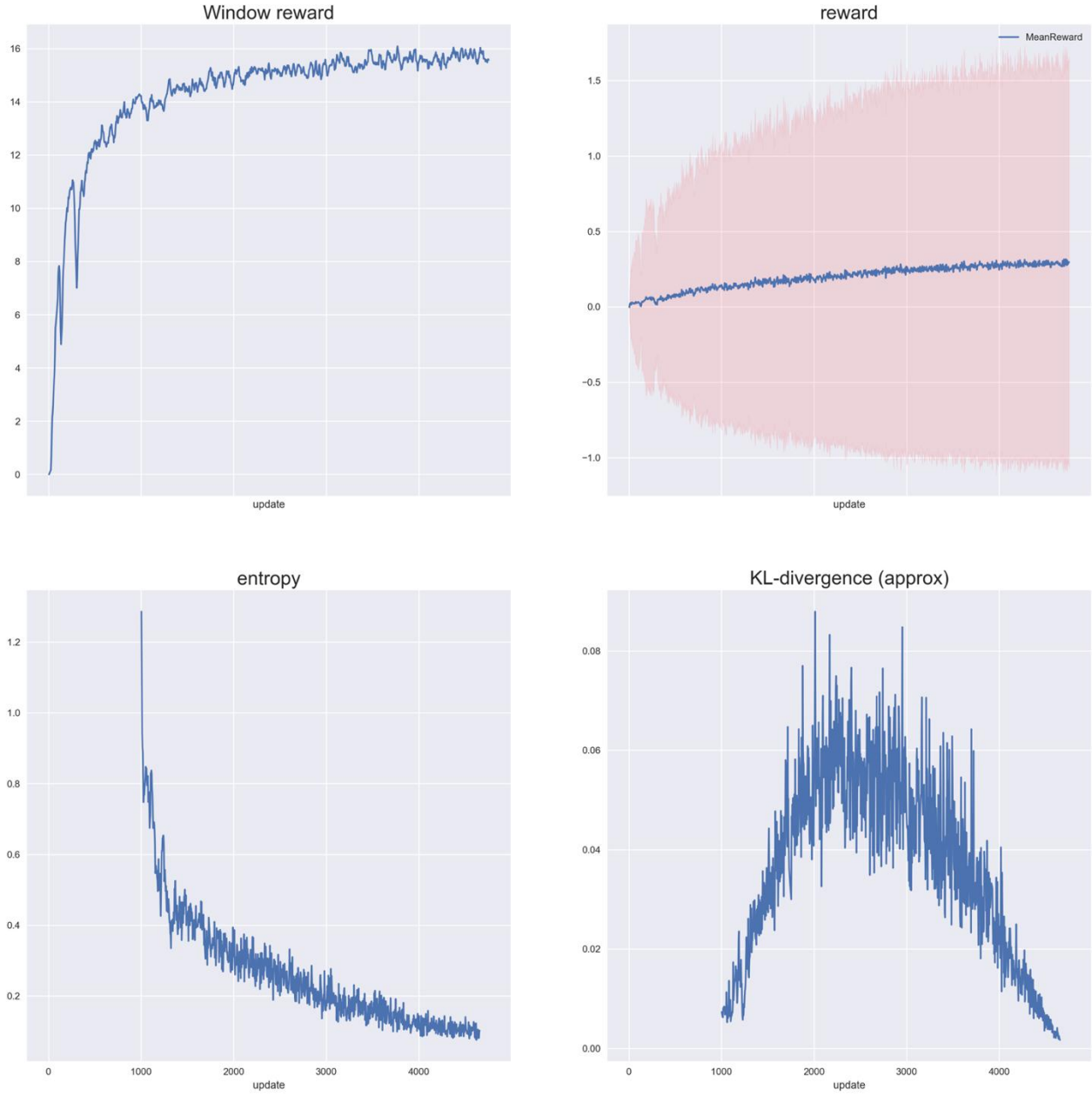


Рисунок 4.2 - Показники найкращої моделі в процесі навчання. Зверху-донизу та справа-наліво: нагорода по вікну, середня нагорода та мінімальна і максимальна нагорода, що зображені червоним, ентропія, апроксимація дивергенції Кульбака-Лейблера

Приведені графіки є досить стандартними для процесу навчання глибоких агентів, що свідчить про відсутність збоїв у його роботі. Ентропія стратегій поступово знижується, що свідчить про поступову збіжність алгоритму, а КЛ-дивергенція спочатку зростає, що свідчить про пожвавлене дослідження в цей час, а потім поступово згасає.

Вдалі траєкторії, якими агент знаходить цільовий об’єкт зображені на рисунку 4.3:

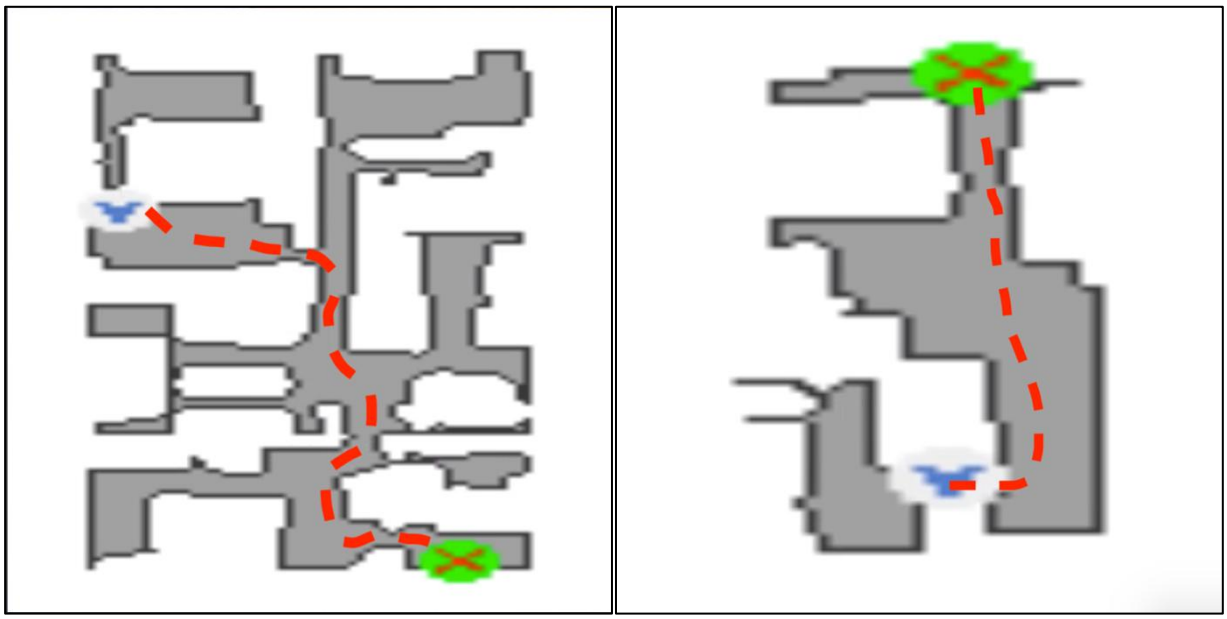


Рисунок 4.3 - Успішні траєкторії моделі на валідаційних сценах

4.2 Основні джерела помилок

Попри те, що агент досить успішно вирішує задачу, його відсоток успішних епізодів (а епізод рахується за успішний, якщо агент спромігся знайти до нього шлях менш ніж за 500 рухів) трохи менший за 1. Основними джерелами помилок агенту є:

- Велика складність шляху та сцени, неспроможність знайти вірний шлях (рисунок 4.4).
- Невеликі перепони, які агент не бачить через висоту посадки камери.
- Недоліки симулятора, який час-від-часу некоректно візуалізує текстури та об’єкти (рисунок 4.5).

Рисунок 4.4 - Збій в роботі алгоритму: агент не знаходить шлях до цілі

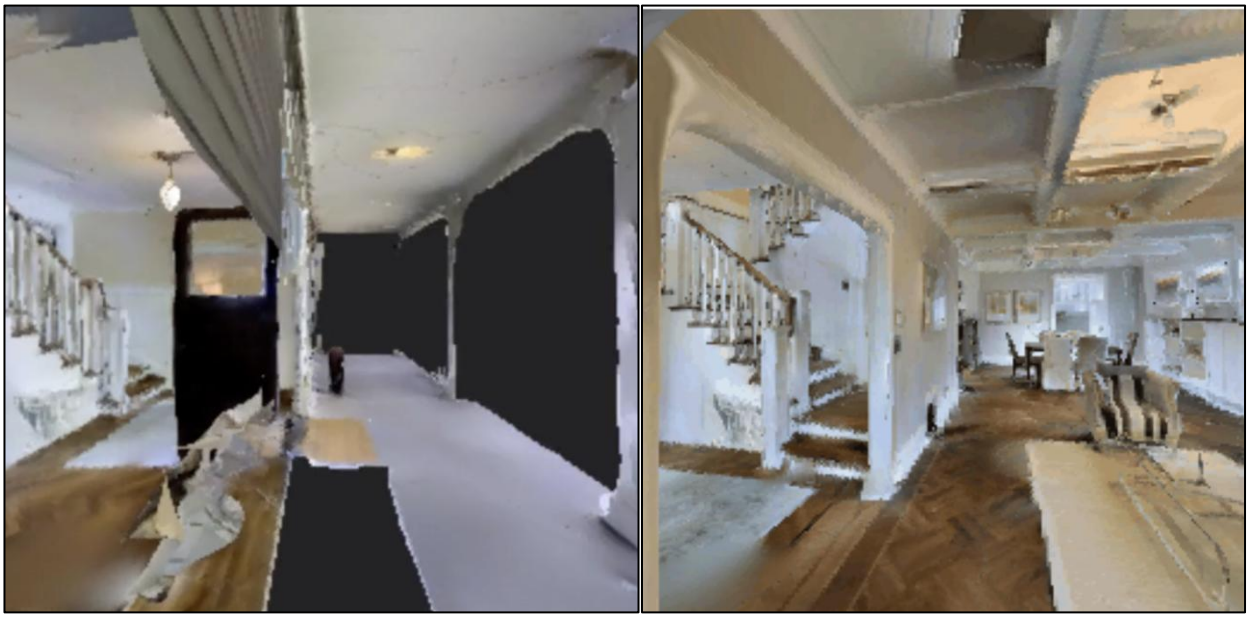


Рисунок 4.5 - Помилки симулятора

Врешті решт, ці недоліки є досить технічними, та можуть вирішуватися додатковою логікою з перевірками та евристиками виходу з таких ситуацій, а тому їх не варто вважати за великі.

4.3 Висновки

В цьому розділі були наведені результати експериментів, що були проведені в рамках цієї роботи, їх опис та деталі, а також графіки із їх порівняльним аналізом. Етап тренування найкращого варіанту був проаналізований, були представлені графіки основних показників, що його характеризують. Приклади траєкторій, побудованих найкращою моделлю також були наведені у вигляді їх візуалізацій.

Також були наведені і проаналізовані основні джерела невеликої кількості помилок, що агент допускає. В результаті детального аналізу було зазначено, що зважаючи на їх невелику кількість та можливість дещо їх вирішити додатковою логікою, всі наведені типи збоїв не викликають великих проблем для експлуатації агенту.

РОЗДІЛ 5 ФУНКЦІОНАЛЬНО – ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі буде проведена оцінка основних характеристик програмного продукту призначеного для симуляції потоку рідин та газів.

Даний продукт розроблений на мові програмування Python з використанням текстового редактору VIM, як окремий модуль. Програмний продукт призначено для використання на будь-яких обчислювальних машинах із системами Linux або OSX, проте рекомендованою вимогою є два графічних процесора NVIDIA, що підтримують CUDA>8.1. Продукт розроблений як кросплатформне програмне забезпечення, тому він працює незалежно від операційної системи (OSX, Linux, Ubuntu, т.д.). Нижче буде наведено аналіз декількох варіантів реалізації модулю з метою вибору найбільш оптимального, враховуючи при цьому як економічні фактори, так і характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

1. Визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На

цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.

2. Для кожної функції визначаються повні річні витрати й кількість робочих годин.
3. Для кожної функції на основі оцінок попереднього пункту визначається кількісна характеристика джерел витрат.
4. Після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

5.1 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічний аналізу розробки.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

– забезпечувати високу швидкість обробки великих об’ємів даних у реальному часі;

– забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використовування його як модуля;

– передбачати мінімальні витрати на впровадження програмного продукту.

5.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного комплексу для навчання та підтримки агентів глибокого навчання з підкріпленням. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

F_1 – вибір мови програмування;

F_2 – вибір інструменту для глибокого навчання;

F_3 – обробка даних одного часового кроку.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

а) мова програмування Python;

б) мова програмування C++;

Функція F_2:

а) бібліотека tensorflow;

б) бібліотека pytorch

Функція F_3:

а) використання вбудованих функцій для мінімізації функцій та дій над матрицями;

б) власна реалізація пакету потрібних функцій.

5.3 Варіанти реалізації основних функцій

Морфологічна карта (рисунок 5.1) відображає всі можливі поєднання варіантів реалізації функцій, які складають повну множину варіантів ПП. На основі карти було побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 5.1).

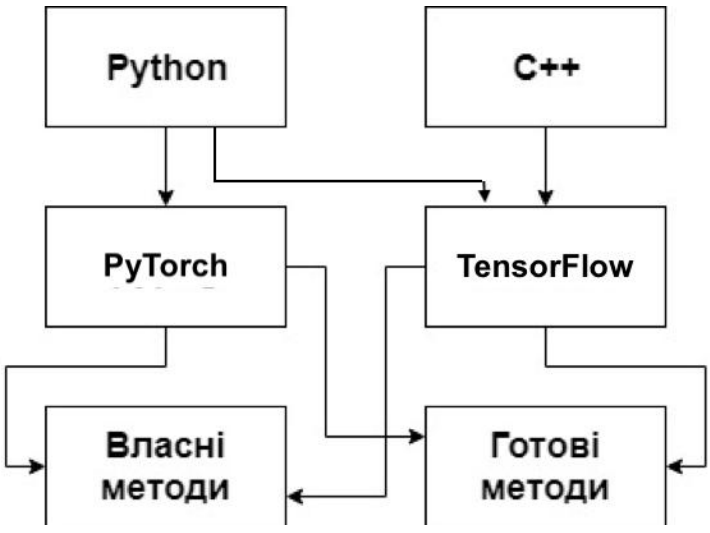


Рисунок 5.1 Морфологічна карта

Таблиця 5.1 – Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
F_1	А	Невелика кількіть коду, зручність проведення експериментів.	Низька швидкодія.
	Б	Висока швидкодія, прикладний підхід до роботи із пам'яттю, можливість оптимізації під різні платформи.	Велика кількість коду.
F_2	А	Простота використання.	Наявні лишні методи, відсутність хорошої документації.
	Б	Містить лише необхідні методи, не потребує додаткового лінкування.	Можуть виникнути складнощі у використанні.
F_3	А	Простота використання. Можлива складність встановлення.	Неможливо додати щось своє, відсутність можливості реалізації оптимізації під різні платорми .
	Б	Використання лише потрібних методів.	Складність реалізації.

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1:

Оскільки в даному висока швидкодія є важливішим фактором, ніж невелика кількість коду, то варіант а) можна не розглядати.

Функція F_2:

Так як простота використання та чистота коду є досить важливими факторами, то варіант а) можна не розглядати.

Так, як наведені варіанти реалізації мають свої значні переваги один над одним, варто розглянути та проаналізувати обидва.

Функція F_3:

Так, як наведені варіанти реалізації мають свої значні переваги один над одним, варто розглянути та проаналізувати обидва.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

1. F_1а – F_2б – F_3а

2. F_1а – F_2б – F_3б

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

5.4 Обґрунтування системи параметрів ПП

5.4.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

X_1 – швидкодія мови програмування;

X_2 – об’єм пам’яті для коректної роботи програми;

X_3 – час обробки кроку;

X_4 – потенційний об’єм програмного коду.

X_1: Відображає швидкодію виконання операцій та методів загалом залежно від обраної мови програмування.

X_2: Відображає необхідний для збереження та обробки даних об’єм оперативної пам’яті пристрою.

X_3: Відображає час, який витрачається на обробку кроку виконання.

X_4: Показує к-ть програмного коду, який необхідно створити розробнику.

5.4.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 5.2.

Таблиця 5.2 – Основні параметри ПП

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	нс/Оп	200	50	1
Об’єм пам’яті для коректної роботи	X2	Мб	32	16	8
Час обробки зображення	X3	мс	2000	840	120
Потенційний об’єм програмного коду	X4	кількість рядків коду	1800	1200	900

За даними таблиці 5.2 будуються графічні характеристики параметрів (рис. 5.2 – рис. 5.5).

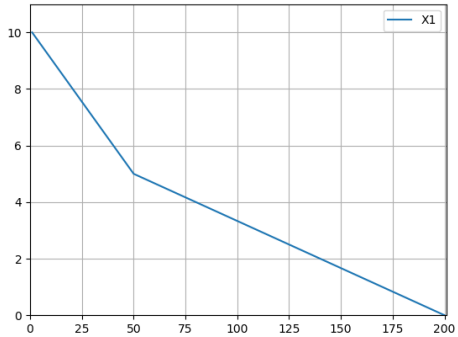


Рисунок 5.2 – X_1, швидкодія мови програмування

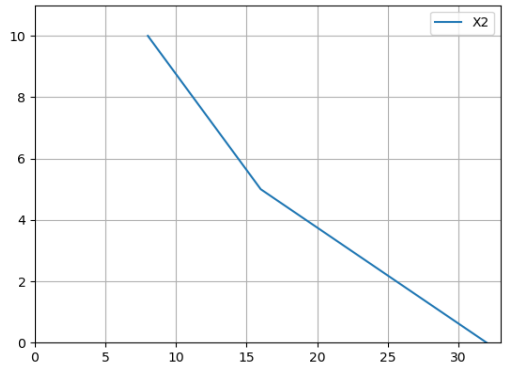


Рисунок 5.3 – X_2, об’єм пам’яті для коректної роботи

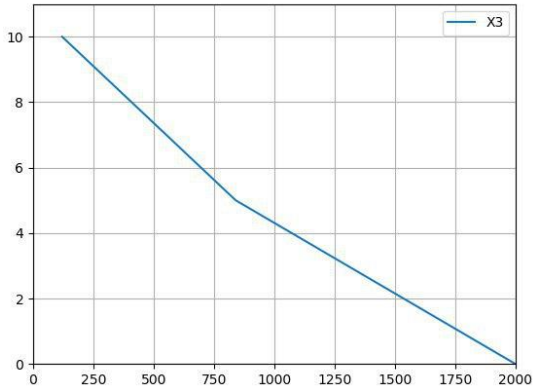


Рисунок 5.4 – X_3, час обробки зображень користувача

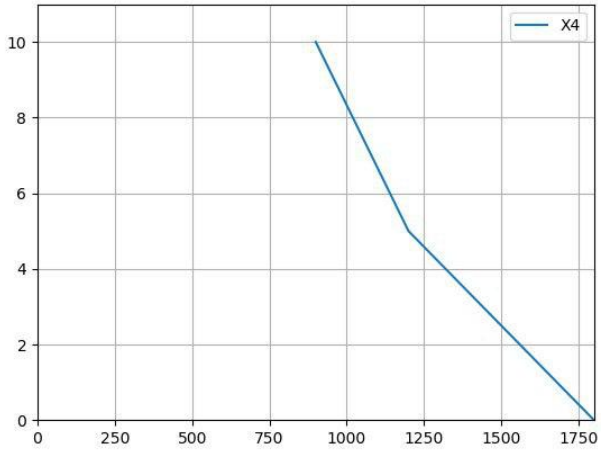


Рисунок 5.5 – X_4, потенційний об’єм програмного коду

5.4.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 5.3.

Таблиця 5.3 – Результати ранжування параметрів

Познач. параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів Ri	Відхи-лення Δi	Δi^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	нс/Оп	4	1	1	1	1	1	2	11	-11	121
X2	Об’єм пам’яті для коректної роботи	Мб	4	4	3	4	4	4	4	27	5	25
X3	Час обробки кроку	Мс	2	2	4	4	4	3	4	23	-1	1
X4	Потенційний об’єм програмного коду	к-сть рядків коду	4	4	3	4	4	4	4	27	5	25
	Разом		14	11	11	13	13	12	14	88	0	172

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70,$$

де N – число експертів,

n – кількість параметрів.

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 22.$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 172.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 172}{7^2(4^3 - 4)} = 0{,}70 > W_k = 0{,}67.$$

Рангування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, який дорівнює 0,67.

Скориставшись результатами рангування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 5.4.

Таблиця 5.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X_1 і X_2	<	<	<	<	<	<	<	<	0,5
X_1 і X_3	<	<	<	<	<	<	<	<	0,5
X_1 і X_4	<	<	<	<	<	<	<	<	0,5
X_2 і X_3	>	=	=	>	=	>	<	<	0,5
X_2 і X_4	<	<	=	=	<	<	<	<	0,5
X_3 і X_4	<	<	=	<	<	<	<	<	0,5

Числове значення, що визначає ступінь переваги i – го параметра над j – тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{matrix} 1.5 \text{ при } X_i > X_j, \\ 1 \text{ при } X_i = X_j, \\ 0.5 \text{ при } X_i < X_j, \end{matrix}$$

З отриманих числових оцінок переваги складемо матрицю $A = \square aij \square$.

Для кожного параметра розрахуємо вагомості Кві за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \text{ де } b_i = \sum_{i=1}^{N} a_{ij}.$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \text{ де } b_i' = \sum_{i=1}^{N} a_{ij} b_j.$$

Як видно з таблиці 5.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 5.5 – Розрахунок вагомості параметрів

Параметри і	Параметри j				Перша ітер.		Друга ітер.		Третя ітер	
	X_1	X_2	X_3	X_4	b_i	K	b_{i1}	$K_{вi1}$	b_{i2}	$K_{вi2}$
X_1	1	0,5	0,5	0,5	2,5	0,156	9,25	0,157	34,125	0,158
X_2	1,5	1	0,5	0,5	3,5	0,219	12,25	0,208	44,875	0,208
X_3	1,5	1,5	1	0,5	4,5	0,281	16,25	0,275	59,125	0,274
X_4	1,5	1,5	1,5	1	5,5	0,344	21,25	0,360	77,875	0,361
Всього:					16	1	59	1	216	1

5.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X_2 (об’єм необхідної оперативної пам’яті) та X_3 (час обробки кроку) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X_1 (швидкодія мови програмування) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 50 нс/Оп або варіанту б) 1 нс/Оп.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 5.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j},$$

де n – кількість параметрів;

$K_{вi}$– коефіцієнт вагомості i – го параметра;

B_i – оцінка i – го параметра в балах.

Таблиця 5.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F_1 (X_1)	А	11000	1,8	0,158	0,2844
$F_2(X_2,$ $X_3)$	А	19	3,7	0,208	0,7696
F3(X_4)	А	760	3,8	0,274	1,0412
	Б	60	5,4	0,361	1,9494

За даними з таблиці 5.6 за формулою

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

$$К_{К1} = 0{,}2844 + 0{,}7696 + 1{,}0412 = 2{,}0952$$
$$К_{К2} = 0{,}2844 + 0{,}7696 + 1{,}9494 = 3{,}0034$$

Як видно з розрахунків, кращим є другий варіант, для якого коефіцієнт технічного рівня має найбільше значення.

5.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як:

$$T_О = T_Р \cdot К_П \cdot К_{СК} \cdot К_М \cdot К_{СТ} \cdot К_{СТ.М},$$

де $T_Р$ – трудомісткість розробки ПП;

$К_П$ – поправочний коефіцієнт;

$К_{СК}$ – коефіцієнт на складність вхідної інформації;

$К_М$ – коефіцієнт рівня мови програмування;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_Р$ = 120 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді, за формулою 5.1, загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 120 \cdot 1.7 \cdot 0.8 = 163.2 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ = 30 людино-днів, $К_П$ = 0.9, $К_{СК}$ = 1, $К_{СТ}$ = 0.8:

$$Т_2 = 30 \cdot 0.9 \cdot 0.8 = 21.6 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$Т_I = (163.2 + 21.6 + 4.8 + 21.6) \cdot 8 = 1689{,}6 \text{ людино-годин;}$$
$$Т_{II} = (163.2 + 21.6 + 6.91 + 21.6) \cdot 8 = 1706{,}48 \text{ людино-годин;}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 14000 грн., один фінансовий аналітик з окладом 12000 грн. Визначимо зарплату за годину за формулою:

$$C_ч = \frac{М}{T_m \cdot t} \text{ грн.,}$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$C_ч = \frac{14000 + 14000 + 12000}{3 \cdot 21 \cdot 8} = 79{,}37 \text{ грн.}$$

Тоді розрахуємо заробітну плату за формулою

$$С_{зп} = С_ч \cdot Т_i \cdot К_Д,$$

де $С_ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП}$ = 79,37· 1689.6 · 1.2 = 160924, 27 грн.

II. $С_{ЗП}$ = 79,37· 1706.48 · 1.2 = 162531, 99 грн.

Відрахування на єдиний соціальний внесок в залежності від групи професійного ризику (II клас) становить 22%:

I. $С_{ВІД} = С_{ЗП}$ · 0.22 = 160924,77*0.22 = 35403,45 грн.

II. $С_{ВІД} = С_{ЗП}$ · 0.22 = 162531,99*0.22 = 35757,04 грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 14000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot M \cdot K_З = 12 \cdot 14000 \cdot 0,2 = 33600 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + K_З) = 33600 \cdot (1 + 0.2) = 40320 \text{ грн.}$$

Відрахування на єдиний соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 40320 * 0,22 = 8870,4 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 25000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 25000 = 7187{,}5 \text{ грн.},$$

де $К_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$ – річна норма амортизації;

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 25000 \cdot 0.05 = 1437{,}5 \text{ грн.},$$

де $К_Р$ – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К - Д_В - Д_С - Д_Р) \cdot t_З \cdot К_В = (365 - 104 - 8 - 16) \cdot 8 \cdot 0.9 = 1706.4 \text{ годин},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В, Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1706{,}4 \cdot 0{,}156 \cdot 0{,}85 \cdot 2{,}7515 = 622.57 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$ – коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 25000 \cdot 0,67 = 16750 \text{ грн.}$$

Тоді річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н$$

$$С_{ЕКС} = 40320 + 8870,4 + 7187,5 + 1437,5 + 622.57 + 16750 = 75187,97 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М-Г} = С_{ЕКС} / \text{ТЕФ} = 75187,97 / 1706,4 = 44,06 \text{ грн/час.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot T$$

I. $С_М = 44,06 * 1689,6 = 74443,7 \text{ грн.};$

II. $С_М = 44,06 * 1706.48 = 75187,5 \text{ грн.};$

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67$$

I. $С_Н = 160924,77 * 0,67 = 107819,6 \text{ грн.};$

II. $С_Н = 162531,99 * 0,67 = 108896,44 \text{ грн.};$

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_{М} + С\_Н$$

I. $С_{ПП}$ = 160924, 27 + 35403,45 + 74443,7 + 107819,6 = 378591 грн.;

II. $С_{ПП}$ = 162531,99 + 35757,04 + 75187,5 + 108896,44 = 382372,97 грн.;

5.7 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР j} = К_{К j} / С_{Ф j},$$

$$К_{ТЕР1} = 2,0952 / 378506,6 = 0,56 \cdot 10\text{-}7;$$

$$К_{ТЕР2} = 3,0034 / 382287,66 = 0,79 \cdot 10\text{-}7;$$

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1}$= 0,79·10-7.

5.8 Висновки

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Загалом сам процес аналізу можна розділити на дві частини.

В першій частині проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу, що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є другий варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості

$$К_{ТЕР} = 0.79 \cdot 10\text{-}7.$$

Цей варіант реалізації програмного продукту має такі параметри:

- мова програмування – Python;
- бібліотека для візуалізації PyTorch;
- власна реалізація пакету потрібних функцій.

Даний варіант виконання програмного комплексу дає користувачу можливість використовувати програмний комплекс глибокого навчання з

підкріпленням з достатньою точністю та швидкодією, а також якмога меншим використанням пам’яті ПК.

ВИСНОВКИ

В цій роботі було проаналізовано існуючі методи для вирішення задачі візуальної навігації та алгоритми глибокого навчання з підкріпленням, спроектовано архітектуру і процедури навчання моделі на базі алгоритму Наближеної оптимізації плану (PPO) із обрізкою. Серед елементів, що варіювалися в різних експериментах, були попереднє навчання імітацією, заохочення агенту до дослідження методом підрахунків із хеш-функцією та методом навчання на базі допитливості.

Було розроблено програмне забезпечення для навчання і валідації моделі, на базі якого проводилися експерименти, що включало в себе програмну реалізацію архітектури моделі та інфраструктури для її навчання з використанням розподілених обчислень на графічних прискорювачах (GPU).

Був проведений детальний аналіз отриманих моделей та їх ефективності в рамках поставленої задачі. Модель, що виявилася найкращою досягла майже бездоганних результатів на валідаційному наборі епізодів, що базувалися на сценах, відсутніх в тренувальному наборі, таким чином продемонструвавши здатність до узагальнення. Також були розібрані джерела помилок, яких допускалася модель, запропоновані методи їх усунення.

У подальшому планується дослідження більш складної задачі, в рамках якої агент отримує команди на природній мові, що описують об’єкт, який він повинен знайти, дослідження задачі, в якій агент шукає відповідь на задане питання у форматі «Чи стоїть стілець на кухні?», «Якого кольору шпалери у вітальні?», а також розробка більш узагальненої бібліотеки для полегшення дослідження цих задач, та впровадження їх в реальні застосунки. Крім того, планується реалізація методів доменного адаптування моделі, що дозволило б пристосувати модель, навчену за допомогою швидкого симулятора до модальності реальних даних із камери та сенсорів-дальномірів, розташованих на фізичному роботі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. E. Rublee, V. Rabaud, K. Konolige, G. Bradski ORB: An efficient alternative to SIFT or SURF. Spain, Barcelona: ICCV, 2011. vol. 11, no. 1, p. 2.
2. D. Navneet, B. Triggs Histograms of oriented gradients for human detection. California, San Diego: CVPR, 2005. vol. 1, pp. 886-893.
3. B. Herbert, T. Tuytelaars, L. Van Gool Surf: Speeded up robust features. Berlin, Heidelberg: European conference on computer vision, 2006. P. 404-417
4. R. Hartley, A. Zisserman Multiple view geometry in computer vision. – Cambridge: Cambridge university press, 2003.
5. A. Krizhevsky, I. Sutskever, G. Hinton Imagenet classification with deep convolutional neural networks. Nevada, Lake Tahoe: Advances in neural information processing systems, 2012. P. 1097-1105
6. K. Simonyan, A. Zisserman Very deep convolutional networks for large-scale image recognition (arXiv preprint arXiv:1409.1556), 2014.
7. K. He, X. Zhang, S. Ren, J. Sun Deep residual learning for image recognition. Nevada, Las Vegas: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016. P. 770-778
8. S. Ren, K. He, R. Girshick, J. Sun Faster r-cnn: Towards real-time object detection with region proposal networks. Canada, Montreal: Advances in neural information processing systems, 2015. P. 91-99
9. K. He, G. Gkioxari, P. Dollár, R. Girshick Mask r-cnn. Hawaii, Honolulu: Proceedings of the IEEE international conference on computer vision, 2017. P. 2961-2969
10. L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, A. Yuille Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. IEEE transactions on pattern analysis and machine intelligence, 2017. P. 834-848
11. O. Ronneberger, P. Fischer, T. Brox U-net: Convolutional networks for biomedical image segmentation. Germany, Munich: International Conference on Medical image computing and computer-assisted intervention, 2015. P. 234-241

12. I. Goodfellow, J. Pouget-Abadie, M. Mirza Generative adversarial nets. Canada, Montreal: Advances in neural information processing systems, 2014. P. 2672-2680
13. H. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger Deep speech: Scaling up end-to-end speech recognition. (arXiv preprint arXiv:1412.5567), 2014.
14. H. Tachibana, K. Uenoyama, S. Aihara Efficiently trainable text-to-speech system based on deep convolutional networks with guided attention. Canada, Calgary: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2018. P. 4784-4788
15. G. Cybenko Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems, 1989. P. 303-314
16. G. Tesauro Temporal difference learning and TD-Gammon. Communications of the ACM, 1995. P. 58-68
17. D. Silver, A. Huang, C. Maddison Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 2014. P. 484.
18. D. Silver, T. Hubert, J. Schrittwieser A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science 362, no. 6419, 2018. P. 1140-1144
19. N. Savinov, A. Dosovitskiy, V. Koltun Semi-parametric topological memory for navigation. (arXiv preprint arXiv:1803.00653), 2018.
20. Japanese Nursing Association. Report on nursing in Japan, 2016. Дата оновлення 03.09.2016. URL: https://www.nurse.or.jp/jna/english/pdf/nursing-in-japan2016.pdf (дата звернення: 09.05.2019).
21. P. Anderson, A. Chang, D. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka On evaluation of embodied navigation agents. (arXiv preprint arXiv:1807.06757), 2018.
22. E. Feinberg, P. Kasyanov, M. Zgurovsky Convergence of value iterations for total-cost mdps and pomdps with general state and action sets. IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), 2014. P. 1-8
23. V. Mnih, A. Badia, M. Mirza Asynchronous methods for deep reinforcement learning. New York: International conference on machine learning, 2016. P. 1928-1937

24. J. Schulman, F. Wolski, P. Dhariwal Proximal policy optimization algorithms. (arXiv preprint arXiv:1707.06347), 2017.
25. J. Schulman, S. Levine, P. Abbeel Trust region policy optimization. France, Lille: International Conference on Machine Learning, 2015. P. 1889-1897
26. T. Lillicrap, J. Hunt, A. Pritzel Continuous control with deep reinforcement learning. (arXiv preprint arXiv:1509.02971), 2015.
27. S. Fujimoto, H. van Hoof, D. Meger Addressing function approximation error in actor-critic methods. (arXiv preprint arXiv:1802.09477), 2018.
28. T. Haarnoja, A. Zhou, P. Abbeel Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor (arXiv preprint arXiv:1801.01290), 2018.
29. V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller Playing atari with deep reinforcement learning. (arXiv preprint arXiv:1312.5602), 2013.
30. M. Bellemare, W. Dabney, R. Munos A distributional perspective on reinforcement learning. Proceedings of the 34th International Conference on Machine Learning-Volume 70, 2017. P. 449-458
31. W. Dabney, M. Rowland, M. Bellemare Distributional reinforcement learning with quantile regression. Second AAAI Conference on Artificial Intelligence, 2018.
32. M. Andrychowicz, F. Wolski, A. Ray Hindsight experience replay. California, Long Beach: In Advances in Neural Information Processing Systems, 2017. P. 5048-5058
33. S. Racanière, T. Weber, D. Reichert Imagination-augmented agents for deep reinforcement learning. California, Long Beach: Advances in neural information processing systems, 2017. P. 5690-5701
34. A. Nagabandi, G. Kahn, R. Fearing Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. IEEE International Conference on Robotics and Automation (ICRA), 2018. P. 7559-7566
35. S. Ross, G. Gordon, D. Bagnell A reduction of imitation learning and structured prediction to no-regret online learning. Proceedings of the fourteenth international conference on artificial intelligence and statistics, 2011. P. 627-635

36. R. Williams Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning 8, no. 3-4, 1992. P. 229-256
37. J. Baxter, P. Bartlett Infinite-horizon policy-gradient estimation. Journal of Artificial Intelligence Research 15, 2001. P. 319-350
38. Z. Kolter, A. Ng Near-Bayesian exploration in polynomial time. Canada, Montreal: Proceedings of the 26th Annual International Conference on Machine Learning, 2009. P. 513-520
39. T. Haoran, R. Houthooft, D. Foote # Exploration: A study of count-based exploration for deep reinforcement learning. California, Long Beach: Advances in neural information processing systems, 2017. P. 2753-2762
40. A. Strehl, M. Littman An analysis of model-based interval estimation for Markov decision processes. Journal of Computer and System Sciences 74, no. 8, 2008. P. 1309-1331
41. D. Pathak, P. Agrawal, A. Efros Curiosity-driven exploration by self-supervised prediction. Hawaii, Honolulu: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2017. P. 16-17
42. A. Sax, B. Emi, A. Zamir Mid-Level Visual Representations Improve Generalization and Sample Efficiency for Learning Active Tasks. (arXiv preprint arXiv:1812.11971), 2018.
43. A. Zamir, A. Sax, W. Shen Taskonomy: Disentangling task transfer learning. Utah, Salt Lake City: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018. P. 3712-3722
44. M. Savva, A. Kadian, O. Maksymets Habitat: A platform for embodied ai research. (arXiv preprint arXiv:1904.01201), 2019.
45. B. Zhou, A. Lapedriza, A. Khosla Places: A 10 million image database for scene recognition. IEEE transactions on pattern analysis and machine intelligence 40, no. 6, 2017. P. 1452-1464
46. E. Feinberg, P. Kasyanov, M. Zgurovsky Partially observable total-cost Markov decision processes with weakly continuous transition probabilities. Mathematics of Operations Research 41, no. 2, 2016. P. 656-681

47. E. Feinberg, P. Kasyanov, N. Zadoianchuk Average cost Markov decision processes with weakly continuous transition probabilities. Mathematics of Operations Research 37, no. 4, 2012. P. 591-607
48. O. Viskov, A. Shiryaev On controls which reduce to optimal stationary regimes. Trudy Matematicheskogo Instituta imeni VA Steklova, 71, 1964. PP. 35-45
49. D. Blackwell Discrete dynamic programming. The Annals of Mathematical Statistics, 1962. P. 719-726

ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО ПРОДУКТУ

```
import torch
import torch.nn as nn

from algo.rl.ppo.utils import Flatten, CategoricalNet
import torchvision.transforms as trn
import torchvision.models as models
import os
from PIL import Image
from torch.autograd import Variable as V

import numpy as np



class Policy(nn.Module):
    def __init__(self,
                 observation_space,
                 action_space,
                 depth_only,
                 use_seg_depth=False,
                 add_scene_classifier=False,
                 hidden_size=512):
        super().__init__()
        self.dim_actions = action_space.n

        self.net = Net(
            depth_only=depth_only,
            add_scene_classifier=add_scene_classifier,
            use_seg_depth=use_seg_depth,
            observation_space=observation_space,
            hidden_size=hidden_size
        )

        self.action_distribution = CategoricalNet(
```

```
            self.net.output_size, self.dim_actions
        )

    def forward(self, *x):
        raise NotImplementedError

    def act(self, observations, rnn_hidden_states, masks, deterministic=False):
        value, actor_features, rnn_hidden_states = self.net(
            observations, rnn_hidden_states, masks
        )
        distribution = self.action_distribution(actor_features)

        if deterministic:
            action = distribution.mode()
        else:
            action = distribution.sample()

        action_log_probs = distribution.log_probs(action)

        return value, action, action_log_probs, rnn_hidden_states

    def get_value(self, observations, rnn_hidden_states, masks):
        value, _, _ = self.net(observations, rnn_hidden_states, masks)
        return value

    def evaluate_actions(self, observations, rnn_hidden_states, masks, action):
        value, actor_features, rnn_hidden_states = self.net(
            observations, rnn_hidden_states, masks
        )
        distribution = self.action_distribution(actor_features)

        action_log_probs = distribution.log_probs(action)
        distribution_entropy = distribution.entropy().mean()

        return value, action_log_probs, distribution_entropy, rnn_hidden_states
```

```
    def evaluate_actions_with_logits(self, observations, rnn_hidden_states, masks, action):
        value, actor_features, rnn_hidden_states = self.net(
            observations, rnn_hidden_states, masks
        )
        distribution = self.action_distribution(actor_features)

        action_log_probs = distribution.log_probs(action)
        distribution_entropy = distribution.entropy().mean()

        return value, action_log_probs, distribution_entropy, rnn_hidden_states,
actor_features



class Normalize(nn.Module):
    def __init__(self, mean, variance):
        self.mean = mean
        self.variance = variance

    def forward(self, x):
        return (x - self.mean) / self.variance



class SceneClassifier(nn.Module):
    def __init__(self):
        super().__init__()

        # th architecture to use
        arch = 'resnet18'

        # load the pre-trained weights
        PATH = os.path.dirname(os.path.abspath(__file__))
        model_dir = os.path.join(PATH, '../../final_checkpoints')
        model_file = os.path.join(model_dir, '%s_places365.pth.tar' % arch)
        if not os.access(model_file, os.W_OK):
            weight_url = 'http://places2.csail.mit.edu/models_places365/' +
'%s_places365.pth.tar' % arch
```

```
            os.system('wget ' + weight_url + ' -P ' + model_dir)


        model = models.__dict__[arch](num_classes=365)

        checkpoint = torch.load(model_file, map_location=lambda storage, loc: storage)

        state_dict = {str.replace(k,'module.',''): v for k,v in
checkpoint['state_dict'].items()}

        model.load_state_dict(state_dict)

        self.model = torch.nn.Sequential(*(list(model.children())[:-1]))

        self.model.eval()


    def transform(self, batch):

        # CenterCrop(224)

        batch = batch[:, 16:240, 16:240, :]


        # ToTensor()

        batch = batch / 255  # normalize

        batch = batch.permute(0, 3, 1, 2)  # NHWC -> NCHW


        # Normalize()

        mean = torch.as_tensor([0.485, 0.456, 0.406],

                               dtype=torch.float32,

                               device=batch.device)

        std = torch.as_tensor([0.229, 0.224, 0.225],

                              dtype=torch.float32,

                              device=batch.device)

        batch.sub_(mean[:, None, None]).div_(std[:, None, None])


        return batch



    def forward(self, observations):

        batch = observations["rgb"]

        batch = self.transform(batch)

        out = self.model.forward(batch)

        return out.view(-1, 512)
```

```
class Net(nn.Module):
    """Network which passes the input image through CNN and concatenates
    goal vector with CNN's output and passes that through RNN.
    """

    def __init__(self,
                 observation_space,
                 hidden_size,
                 depth_only,
                 use_seg_depth=False,
                 add_scene_classifier=False):
        super().__init__()

        self._n_input_goal = observation_space.spaces["pointgoal"].shape[0]
        self._hidden_size = hidden_size
        self.depth_only = depth_only
        self.use_seg_depth = use_seg_depth
        self.seg_model = None
        self.scene_classifier = None
        if add_scene_classifier:
            self.scene_classifier = SceneClassifier()

        self.cnn = self._init_perception_model(observation_space)

        if self.is_blind:
            self.rnn = nn.GRU(self._n_input_goal, self._hidden_size)
        else:
            self.rnn = nn.GRU(
                self.output_size + self._n_input_goal, self._hidden_size
            )

        self.critic_linear = nn.Linear(self._hidden_size, 1)
        # self.feature_aligner = nn.Linear(self.output_size +
        #                                  (512 if add_scene_classifier else 0),
        #                                  self.output_size)
```

```
    if add_scene_classifier:
        self.feature_aligner = nn.Conv2d(self.output_size +
                                         (512 if add_scene_classifier else 0),
                                         self.output_size, kernel_size=1)
    else:
        self.feature_aligner = None

    self.layer_init()
    self.train()

def _init_perception_model(self, observation_space):
    if "rgb" in observation_space.spaces and not self.depth_only:
        self._n_input_rgb = observation_space.spaces["rgb"].shape[2]
        if self.use_seg_depth and "depth" in observation_space.spaces:
            self.seg_model = load_model("ade20k")
            self._n_input_rgb = 10
    else:
        self._n_input_rgb = 0

    if "depth" in observation_space.spaces:
        self._n_input_depth = observation_space.spaces["depth"].shape[2]
    else:
        self._n_input_depth = 0

    # kernel size for different CNN layers
    self._cnn_layers_kernel_size = [(8, 8), (4, 4), (3, 3)]

    # strides for different CNN layers
    self._cnn_layers_stride = [(4, 4), (2, 2), (1, 1)]

    if self._n_input_rgb > 0:
        cnn_dims = np.array(
            observation_space.spaces["rgb"].shape[:2], dtype=np.float32
        )
    elif self._n_input_depth > 0:
```

```
    cnn_dims = np.array(
        observation_space.spaces["depth"].shape[:2], dtype=np.float32
    )

if self.is_blind:
    return nn.Sequential()
else:
    for kernel_size, stride in zip(
        self._cnn_layers_kernel_size, self._cnn_layers_stride
    ):
        cnn_dims = self._conv_output_dim(
            dimension=cnn_dims,
            padding=np.array([0, 0], dtype=np.float32),
            dilation=np.array([1, 1], dtype=np.float32),
            kernel_size=np.array(kernel_size, dtype=np.float32),
            stride=np.array(stride, dtype=np.float32),
        )

    return nn.Sequential(
        # Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),
        nn.Conv2d(
            in_channels=self._n_input_rgb + self._n_input_depth,
            out_channels=32,
            kernel_size=self._cnn_layers_kernel_size[0],
            stride=self._cnn_layers_stride[0],
        ),
        nn.ReLU(),
        nn.Conv2d(
            in_channels=32,
            out_channels=64,
            kernel_size=self._cnn_layers_kernel_size[1],
            stride=self._cnn_layers_stride[1],
        ),
        nn.ReLU(),
        nn.Conv2d(
            in_channels=64,
```

```
                out_channels=32,
                kernel_size=self._cnn_layers_kernel_size[2],
                stride=self._cnn_layers_stride[2],
            ),
            Flatten(),
            nn.Linear(32 * cnn_dims[0] * cnn_dims[1], self._hidden_size),
            nn.ReLU(),
        )

def _conv_output_dim(
    self, dimension, padding, dilation, kernel_size, stride
):
    """Calculates the output height and width based on the input
    height and width to the convolution layer.

    ref: https://pytorch.org/docs/master/nn.html#torch.nn.Conv2d
    """
    assert len(dimension) == 2
    out_dimension = []
    for i in range(len(dimension)):
        out_dimension.append(
            int(
                np.floor(
                    (
                        (
                            dimension[i]
                            + 2 * padding[i]
                            - dilation[i] * (kernel_size[i] - 1)
                            - 1
                        )
                        / stride[i]
                    )
                    + 1
                )
            )
```

```
        )
    return tuple(out_dimension)

@property
def output_size(self):
    return self._hidden_size

def layer_init(self):
    for layer in self.cnn:
        if isinstance(layer, (nn.Conv2d, nn.Linear)):
            nn.init.orthogonal_(
                layer.weight, nn.init.calculate_gain("relu")
            )
            nn.init.constant_(layer.bias, val=0)

    for name, param in self.rnn.named_parameters():
        if "weight" in name:
            nn.init.orthogonal_(param)
        elif "bias" in name:
            nn.init.constant_(param, 0)

    nn.init.orthogonal_(self.critic_linear.weight, gain=1)
    nn.init.constant_(self.critic_linear.bias, val=0)

    if self.scene_classifier:
        nn.init.orthogonal_(self.feature_aligner.weight, gain=1)
        nn.init.constant_(self.feature_aligner.bias, val=0)

def forward_rnn(self, x, hidden_states, masks):
    if x.size(0) == hidden_states.size(0):
        x, hidden_states = self.rnn(
            x.unsqueeze(0), (hidden_states * masks).unsqueeze(0)
        )
        x = x.squeeze(0)
        hidden_states = hidden_states.squeeze(0)
    else:
```

```
# x is a (T, N, -1) tensor flattened to (T * N, -1)
n = hidden_states.size(0)
t = int(x.size(0) / n)

# unflatten
x = x.view(t, n, x.size(1))
masks = masks.view(t, n)

# steps in sequence which have zero for any agent. Assume t=0 has
# a zero in it.
has_zeros = (
    (masks[1:] == 0.0).any(dim=-1).nonzero().squeeze().cpu()
)

# +1 to correct the masks[1:]
if has_zeros.dim() == 0:
    has_zeros = [has_zeros.item() + 1]  # handle scalar
else:
    has_zeros = (has_zeros + 1).numpy().tolist()

# add t=0 and t=T to the list
has_zeros = [0] + has_zeros + [t]

hidden_states = hidden_states.unsqueeze(0)
outputs = []
for i in range(len(has_zeros) - 1):
    # process steps that don't have any zeros in masks together
    start_idx = has_zeros[i]
    end_idx = has_zeros[i + 1]

    rnn_scores, hidden_states = self.rnn(
        x[start_idx:end_idx],
        hidden_states * masks[start_idx].view(1, -1, 1),
    )
```

```
                outputs.append(rnn_scores)

            # x is a (T, N, -1) tensor
            x = torch.cat(outputs, dim=0)
            x = x.view(t * n, -1)  # flatten
            hidden_states = hidden_states.squeeze(0)

        return x, hidden_states

    @property
    def is_blind(self):
        return self._n_input_rgb + self._n_input_depth == 0

    def forward_perception_model(self, observations):
        cnn_input = []
        if self._n_input_rgb > 0:
            rgb_observations = observations["rgb"]
            # permute tensor to dimension [BATCH x CHANNEL x HEIGHT X WIDTH]
            rgb_observations = rgb_observations.permute(0, 3, 1, 2)
            if self.use_seg_depth:
                assert self._n_input_depth > 0, "Cant invoke seg_depth without depth =("
                depth_observations = observations["depth"]
                # permute tensor to dimension [BATCH x CHANNEL x HEIGHT X WIDTH]
                depth_observations = depth_observations.permute(0, 3, 1, 2)
                batch = torch.nn.functional.upsample_bilinear(rgb_observations, size=(513,
513))
                batch = batch.cpu().numpy().astype(np.uint8)
                seg_map = self.seg_model.infer(batch.transpose(0, 2, 3, 1))
                mask = (seg_map == np.arange(0, 10)[:, None, None,
None]).astype(int).transpose(1, 0, 2, 3)
                mask = torch.FloatTensor(mask)
                device = torch.device(depth_observations.device)
                mask = torch.nn.functional.upsample_bilinear(mask, size=(256,
256)).to(device)
                seg_depth_observations = mask * depth_observations
                cnn_input.append(seg_depth_observations)
            else:
```

```
                rgb_observations = rgb_observations / 255.0  # normalize RGB
                cnn_input.append(rgb_observations)

        if self._n_input_depth > 0:
            depth_observations = observations["depth"]
            # permute tensor to dimension [BATCH x CHANNEL x HEIGHT X WIDTH]
            depth_observations = depth_observations.permute(0, 3, 1, 2)
            cnn_input.append(depth_observations)

        cnn_input = torch.cat(cnn_input, dim=1)

        return self.cnn(cnn_input)

    def forward(self, observations, rnn_hidden_states, masks):
        x = observations["pointgoal"]

        if not self.is_blind:
            embed = self.forward_perception_model(observations)
            if self.scene_classifier:
                with torch.no_grad():
                    scene_classifier_embed = self.scene_classifier.forward(observations)
                embed = torch.cat([scene_classifier_embed, embed], dim=1)

                n_feats = embed.shape[1]
                embed = self.feature_aligner(embed.view(-1, n_feats, 1, 1)).squeeze(-
1).squeeze(-1)
            x = torch.cat([embed, x], dim=1)
        x, rnn_hidden_states = self.forward_rnn(x, rnn_hidden_states, masks)

        return self.critic_linear(x), x, rnn_hidden_states
```

ДОДАТОК Б ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДЛЯ ДОПОВІДІ

Система прийняття рішень для навігації в середовищах з неповною інформацією

Виконав:
Студент 4-го курсу
Групи КА-53
Титаренко А.М.

Керівник:
д-р фіз.-мат. наук, професор
Касьянов П.О.

Дослідження

- Об’єкт дослідження: Задача візуальної навігації
- Предмет дослідження: Застосування методів глибокого навчання з підкріпленням для вирішення задачі візуальної навігації
- Мета дослідження: Розробка та навчання системи прийняття рішень для візуальної навігації

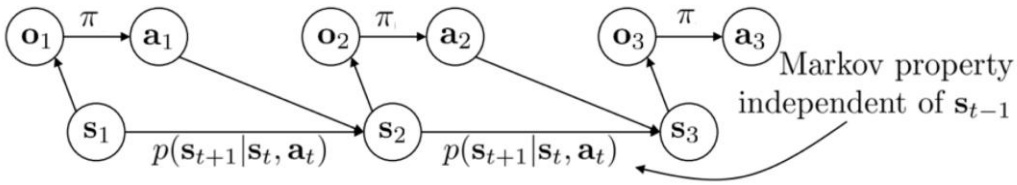
Актуальність

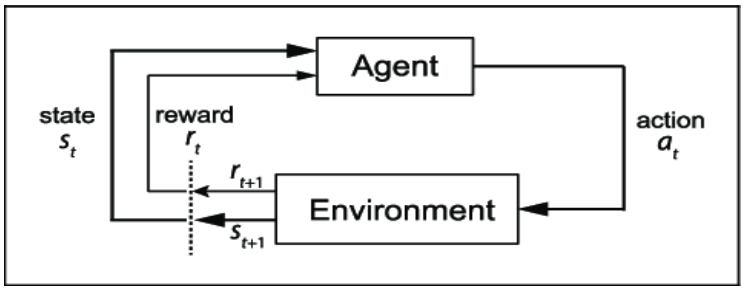
- Задача візуальної навігації наразі є предметом активних досліджень в машинному навчанні, комп’ютерному зорі та штучному інтелекті в цілому
- Якість рішення задачі впливає на розвиненість автономних роботизованих систем, що виконують доручення користувача у фізичному світі.
- Майже повна відсутність рішень на базі навчання з підкріпленням, що тільки набирає популярності через розвиток обчислювальної техніки та глибокого навчання

Постановка задачі

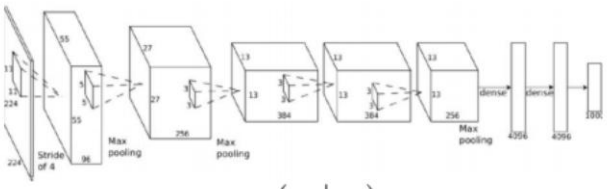
- Розробити систему для навчання та підтримки вибраних алгоритмів глибокого навчання з підкріпленням.
- Навчити агент, який на кожному кроці за візуальним сигналом (Зображення з камери та карта глибин) та інформацією про положення цілі (кут та відстань) приймав би рішення щодо руху, які б приводили до знаходження цілі за коротший час.
- Упевнитися в здатності агенту до пошуку в середовищах, відсутніх в наборі для тренування.

Модель взаємодії середовища і агенту





Глибоке навчання з підкріпленням



$\mathbf{o}_t$ $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ $\mathbf{a}_t$

$\mathbf{s}_t$ – стан
$\mathbf{o}_t$ – спостереження
$\mathbf{a}_t$ – дія

$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ – функція-
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ – план

Цільовий функціонал

$$\pi_\theta(\tau) = \pi_\theta(s_1, a_1, \dots, s_T, a_T) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$$

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[R(\tau)]$$

$$J(\theta) = \int \pi_\theta(\tau) R(\tau) d\tau$$

$$Q(s_t, a_t) = \sum_{t'=t}^{T} \mathbb{E}_{\pi_\theta}[R(s_{t'}, a_{t'})|s_t, a_t]$$

$$V(s_t) = \mathbb{E}_{a_t \sim \pi_\theta(a_t, s_t)}[Q(s_t, a_t)]$$

Існування оптимальної стратегії

- **Теорема** Для Марковських процесів прийняття рішень із скінченними просторами дій та станів існує оптимальна марковська інформаційна стратегія, яка максимізує цільовий функціонал (очікувану кумулятивну винагороду)

Пошук оптимальної стратегії методом градієнту стратегії

$$\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau) R(\tau) d\tau = \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) R(\tau) d\tau =$$

$$= \mathbb{E}_{\tau \sim \pi_\theta(\tau)} [\nabla_\theta \log \pi_\theta(\tau) R(\tau)]$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t | s_t) \right) \left(\sum_{t=1}^{T} R(s_t, a_t) \right) \right]$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t} | s_{i,t}) \right) \left(\sum_{t=1}^{T} R(s_{i,t}, a_{i,t}) \right) \right]$$

Алгоритм REINFORCE

Ініціалізувати довільними значеннями параметри стратегії π_θ

Повторювати:

Генерування набору прикладів, використовуючи стратегію $\pi_\theta(a_t|s_t)$

$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} [(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t))(\sum_{t=1}^{T} R(s_t, a_t))]$

$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Зниження дисперсії

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t} | s_{i,t}) \left(\sum_{t\prime=t}^{T} R(s_{i,t\prime}, a_{i,t\prime}) \right) \right]$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} [\nabla_\theta \log \pi_\theta(\tau) [R(\tau) - b]]$$

Актор-Критик

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(a_{i,t}|s_{i,t}\right) A^{\pi_\theta}\left(s_{i,t}, a_{i,t}\right)\right]$$

$$A^{\pi_\theta}\left(s_{i,t}, a_{i,t}\right) = Q\left(s_{i,t}, a_{i,t}\right) - V(s_{i,t})$$

Наближення

$$A^{\pi}(s_t, a_t) \approx R(s_t, a_t) + V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$$

Алгоритм Наближеної оптимізації стратегії (PPO)

$$\mathbb{E}_{(s_t,a_t)\sim p_\theta(s_t,a_t)}\left[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} A^{\pi_\theta}(s_t, a_t)\right] \rightarrow max$$

При обмеженні:

$$D_{KL}(\pi_{\theta'}(a_t|s_t)\|\pi_\theta(a_t|s_t)) \leq \epsilon$$

Ідея: Робити невеликі кроки від плану до плану